C-50-657
#3

Final Report for Period: 07/2001 - 06/2006
Submitted on: 08/07/2006
Principal Investigator: Tovey, Craig A.
Award ID: 0098807
Organization: GA Tech Res Corp - GIT
Title:
Understanding and Improving On-Line Planning Methods

Project Participants

Senior Personnel

Name: Koenig, Sven
Worked for more than 160 Hours: Yes
Contribution to Project:

Name: Tovey, Craig
Worked for more than 160 Hours: Yes
Contribution to Project:

Name: Tovey, Craig
Worked for more than 160 Hours: Yes
Contribution to Project:

Post-doc

Graduate Student

Name: Furcy, David
Worked for more than 160 Hours: Yes
Contribution to Project:

Name: Vroon, Daron
Worked for more than 160 Hours: Yes
Contribution to Project:

Name: Greenberg, Samuel
Worked for more than 160 Hours: Yes
Contribution to Project:

Name: Likhachev, Maxim
Worked for more than 160 Hours: Yes
Contribution to Project:

Name: Mudgal, Apurva
Worked for more than 160 Hours: Yes
Contribution to Project:

Name: Liu, Yaxin
Worked for more than 160 Hours: Yes
Contribution to Project:
Completed research and dissertation on risk-sensitive planning, including as a highlighted application planners for robot motion under uncertainty in the cases of risk-averse exponential utilities and risk-seeking exponential utilities.

Undergraduate Student

Name: Berhault, Marc
Worked for more than 160 Hours: No
Contribution to Project:

Technician, Programmer

Other Participant

Name: Svennebring, Jonas
Worked for more than 160 Hours: Yes
Contribution to Project:

Research Experience for Undergraduates

Organizational Partners

University of Southern California

SUNY at Stoneybrook

Microsoft Research
Collaboration included co-authorship on papers.

UCLA Computer Science

IBM Yorktown Heights
Yaxin Liu interned at IBM and later was supported at Georgia Tech by an IBM fellowship. Richard Goodwin of IBM Yorktown Heights was a member of the Ph.D. committee and co-author on research papers.

Other Collaborators or Contacts

Ivan Chase SUNY Stony Brook Biologist Discussed animal group hierarchy formation methods to mimic for robot teams.

Joe Mitchell Computer Scientist Advice on computational geometry
issues that arise with 360 degree robot laser sensors.

Anthony Stenz, Carnegie-Mellon Roboticist Research collaborator and co-author.

Activities and Findings

Research and Education Activities:
Research Goal

Search in robotics tends to differ from classical search in several ways. The search domain, even its structure, is only partially known at the outset. More information can be obtained incrementally, but at a real cost of robot motion and time. New information triggers replanning. Thus the search and the search plan are inextricably intertwined.

The main research goal of the project is understand and improve methods for this kind of search-and-replanning. This involves finding effective ways to interleave movement, information acquisition, and planning, and to find synergies and effective tradeoffs among these three necessary activities. In addition, the information acquisition methodology may need to decide, on the fly, how the information is to be structured.

Activities

We have conducted analysis and development of planning methods for robots in which information becomes available during the course of the plan execution. To ensure relevance, we have focused on fundamental tasks such as motion planning, goal-seeking and mapping, and on algorithms that are in current use by real robots.

The principal methodologies we have employed are algorithm analysis, experimentation and simulation. In cases where the reliability of robot actuation and sensing is high or well-understood, simulation has been preferred. In the case of ant-inspired trail-laying robots, laboratory experimentation was employed as well. We have analyzed algorithms for performance guarantees and worst-case behavior, and tested them empirically to evaluate their typical performance. The analysis and testing often reveal the essence of what makes a method effective, or the circumstances that lead it to perform well or poorly. These insights, in turn, lead to improvements of various types, including the following: faster ways to find plans of equivalent or near-equivalent quality; algorithms to find better quality plans; hybrids of two planning methods that inherit the principal advantages of each; general principles for effective on-line planning.

We have also sought general insights and principles that can be derived from the research results, and we have preliminarily explored the application of these principles to robot team planning. The part of the team planning research that is auction-based has evolved into a separate project in its own right. We have also investigated how search methods may take risk attitudes into account.

Several graduate students, as well as an undergraduate and a postdoc, were trained in the course of this project. Their education was distinctive in its blend of computing disciplines; the CS theory students interacted with AI personnel, and the AI and robotics students interacted with theory personnel.

Details on project activities are given in the sections below, in reverse chronological order.

YEAR 3

In the third year of the project, we have continued the research from the first two years. We derived improved bounds for localization, refined the localization method that we started to develop in the second year, and - for the first time - studied fast motion planning algorithms that can take kinematic constraints into account.

MOTION PLANNING:

Motion planning involves finding trajectories in high-dimensional continuous configuration spaces, for example, by using discrete search methods after discretizing the configuration spaces. Configuration spaces can be discretized in different ways, for example with roadmap or cell-decomposition techniques:

Roadmap techniques determine graphs that lie in freespace and represent its connectivity. Systematic techniques are not well suited for high-dimensional spaces. An example is techniques that construct Voronoi graphs. Consequently, researchers use sampling techniques. An example is rapidly exploring random trees (RRTs), a simple but versatile roadmap technique that builds trees.

Sampling techniques are typically probability-complete, meaning that they find a trajectory, if one exists, with a probability that approaches one as their run time increases.

Cell-decomposition techniques, on the other hand, decompose the configuration space into cells. They are typically systematic and thus resolution-complete, meaning that they find a trajectory if one exists within the minimum resolution of the decomposition. Uniform terrain discretizations can prevent one from finding a plan if they are too coarse-grained and result in large spaces that cannot be searched efficiently if they are too fine-grained. Consequently, researchers use nonuniform terrain discretizations. An example is the parti-game method, a reinforcement-learning method that starts with a coarse terrain discretization and refines it during execution by splitting cells only when and where it is needed (for example, around obstacles).

We now proposed a novel technique that combines the advantages of RRTs and the parti-game method. Our parti-game directed RRTs (PDRRTs) are based on the parti-game method but use RRTs as local controllers. PDRRTs differ from recent work that studied hybrids of two different sampling techniques, such as RRTs and probabilistic roadmaps, because they provide a systematic way of improving the performance of RRTs. Our main insight is precisely that the combination of sampling and systematic techniques can result in very powerful motion-planning techniques.

Depending on their parameters, PDRRTs can behave like RRTs, the parti-game method, or a hybrid. Our experimental results show that PDRRTs can plan faster and solve more motion-planning problems than RRTs because the parti-game method directs the searches performed by the RRTs, which allows PDRRTs to solve more motion-planning problems with small passages. Our results also show that PDRRTs can plan faster and with less memory than the parti-game method because RRTs are more capable controllers than the simplistic controllers used by the parti-game method, which allows PDRRTs to split fewer cells than the parti-game method.

LOCALIZATION:

We continued our development of a localization algorithm from year 2, finding a strongly polynomial time O(log^2 n log r)-factor approximation algorithm on polygons, where r is the number of reflex vertices. Extending the

algorithm to apply to continuous polygonal domain required a new hypothesis

equivalence decomposition of the plane built from pairs of aspect graph duals. From a theoretical point of view, our new localization algorithm is a significant breakthrough. We are currently in the process of submitting the result to a conference. In the coming year, we will seek to enhance the practical impact of our research in two ways. First, we will explore computationally practical variations of the algorithm for domains with no sensor or actuator uncertainty. Second, we will explore extensions of the algorithm (and new algorithms) for domains with sensor and actuator uncertainty, as required for actual implementations on mobile robots.

MAPPING:

We also refined our analysis of greedy mapping, which moves the robot from its current location on a shortest path towards a closest useful location, until the

terrain is mapped. We identified four different versions of Greedy Mapping: nearest unvisited, nearest unscanned, nearest unscanned with replanning, and nearest informative. For each version, although the analysis required sometimes differed, we reduced the upper bound on the number of movements of greedy mapping from O(|V|^3/2) to O(|V| ln |V|) edge traversals, where |V| is the number of vertices of the graph that discretized the terrain. This upper bound demonstrates that the travel distance of greedy mapping is guaranteed to be small and thus suggests that greedy mapping is indeed a reasonable mapping method. Moreover, the guaranteed good performance of greedy mapping is robust in that it holds for the different versions of greedy mapping, regardless of sensor type and sensor range.

YEAR 2

In the second year of the project, we have continued the research that we started in the first year and, additionally, investigated a quite surprising application of on-line planning methods, namely covering terrain repeatedly with ant robots. Sven Koenig also gave tutorials at the International Conference on Artificial Intelligence Planning and Scheduling (AIPS), the National Conference on Artificial Intelligence (AAAI), and the International Conference on Robotics and Automation (ICRA) in 2002. These tutorials contained, among other material, our findings from the first year of the project.

MAPPING:

In the previous year, we showed that the worst-case travel distance of D* is at least on the order of (log |V| / log log |V|) |V| steps for graphs with |V| vertices, even if the graphs are planar. We also showed that it is at most on the order of |V|^(3/2) steps. This year, we have derived tighter results. In particular, we can now show that the worst-case travel distance of D* is at least on the order of (log |V| / log log |V|) |V| steps for grid graphs with |V| vertices. This establishes that D* has a super-linear worst-case travel distance on the class of graphs used on real robots. Furthermore, we can now show that the worst-case travel distance of D* is at most on the order of |V| log |V| for planar graphs. This leaves only a log log |V| gap, and establishes that the travel distance of D* is only slightly worse than the travel distance of depth-first search in this worst-case sense. We can now also show that the worst-case travel distance of D* is at most on the order of |V| log^2 |V| on arbitrary graphs, and D* therefore provides a rather good performance guarantee in general.

LOCALIZATION:

In the previous year, we showed that it is NP-hard to find a localization plan whose worst-case travel distance is within a logarithmic factor of minimum. We also studied the performance of a simple greedy localization method. This year, we developed a more sophisticated localization method, by reducing the localization problem to a series of directed Steiner tree-like problems, in which the objective is to find a minimum cost tree rooted at the robot's current location which covers at least half of the Steiner nodes. Using an approximation method for the latter problem, we derive a localization algorithm with a runtime of at most on the order of |V|^{3 log |V|} and a worst-case travel performance ratio of at most on the order of log^3 |V|. This is the first localization algorithm discovered with nearly polynomial run time and high quality performance ratio (compare log^3|V| with for example the ratios |V| and |V|/log |V| that we found in the first year for two simple heuristics).

ANT ROBOTS:

Autonomous agents must be able to make good decisions in complex situations that involve a substantial degree of uncertainty, yet find solutions in a timely manner despite a large number of potential contingencies. Thus, one needs planning techniques that speed up planning by

sacrificing the optimality of the resulting plans. Greedy on-line planning interleaves planning and plan execution to allow agents to gather information early and then uses the acquired information right away for replanning, which reduces the amount of planning performed for unencountered situations and thus allows agents to plan efficiently and be reactive to their current situation.

Agent-centered search is an example of greedy on-line planning. It restricts planning to the part of the domain around the current state of an agent (local search). The part of the domain around the current state of the agent is the part of the domain that is immediately relevant for it in its current situation because it contains the states that it will soon be in. Agent-centered search methods do not plan all the way from the start state to a goal state. Instead, they decide on the local search space, search it, and determine which actions to execute within it to gain information quickly. Then, they execute these actions (or only the first action) and repeat the overall process from their new state, until the planning task is solved.

Real-time search methods are agent-centered search methods that are able to make decisions in any given amount of time. This is, for example, important for applications in interactive entertainment, where computer-controlled characters have to move smoothly. This year, we have investigated a quite surprising application of real-time search methods, namely covering terrain repeatedly with ant robots.

We say that a team of robots covers terrain repeatedly if and only if every location is swept by the body of some robot repeatedly. Dynamic coverage is an important task in mobile robotics, for example in the context of mine sweeping, surveillance, surface inspection, cleaning hazardous waste, and guarding terrain. Consequently, researchers have developed many coverage methods. Most of these methods make the unrealistic assumption that the robots know their location with certainty. The currently popular POMDP-based robot architectures attempt to overcome this problem by providing robots with the best possible location estimates. However, this approach is complicated and can be brittle for robots that are small and cheap and thus have extremely noisy actuators and sensors. We therefore explore an alternative, namely trail-laying and trail-following robots (= ant robots). Our inspiration comes from researchers who studied ant robots that lay trails and follow the trails laid by other ant robots, similar to ants that lay and follow pheromone trails. Ant robots that follow trails arrive at their destinations without having to know their exact locations, which eliminates solving difficult and time-consuming localization tasks. They need only simple sensors, namely sensors that are able to sense the trails, which are artificial landmarks that can be carefully designed to simplify sensing. We utilize a similar idea to build ant robots that cover terrain repeatedly without knowing where they are in the terrain. Similar to ant robots that lay and follow trails, they only need to leave trails in the terrain and sense trails in their neighborhood. Different from ant robots that follow trails, however, they need to move away from trails rather then follow them (to cover terrain that they have not yet covered or not recently covered). We are interested in both single ant robots and teams of ant robots. Teams of ant robots have the potential to cover terrain faster and be more fault tolerant than single ant robots if ant robots can fail. The ant robots do not need to communicate with each other except via the trails, which coordinate the ant robots implicitly and allow them to cover terrain faster than without any communication.

The navigation behavior of our ant robots is controlled by real-time search methods. They robustly cover terrain even if they do not have any memory, do not know the terrain, cannot maintain maps of the terrain, nor plan complete paths. In particular, they cover terrain even if some ant robots fail, they are moved without realizing this (say, by people running into them and pushing them accidentally to a different location), the trails are of uneven quality, or some trails are destroyed. Our main research contribution is to show the abilities of robots with very limited sensing, processing, and communication capabilities. Another research contribution is to show that real-time search methods can be used to implement such robots, which provides a solid theoretical foundation for our approach.

YEAR 1

In the first year of the project, we concentrated on studying and advancing the basic theory of mapping and localization methods. In both of these cases, robots cannot predict with certainty which observations they will make after they have moved. We recruited graduate students to work on the project, and brought them up to speed. They have been active contributors to the research.

MAPPING:

We studied robot navigation tasks in initially unknown terrain, both acquiring a map (mapping) and navigating to a given goal location (goal-directed navigation). In particular, we studied two different planning methods that have successfully been used on mobile robots but not been analyzed before. The planning methods differ both in the technique they use to speed up planning and in the robot-navigation task they solve. Greedy mapping (GM) uses agent-centered search to map unknown terrain. Stentz' Dynamic A* (D*) uses assumption-based planning to navigate to given goal coordinates in unknown terrain. The reason for using these methods is that they make planning in nondeterministic domains tractable and thus result in small planning times. However, it is also important to understand how good the resulting travel distances of the robots and thus the plan-execution times are, since both the planning and plan-execution times together determine the overall performance

of the robot.

In the cases of GM and D*, each planning episode requires solving a shortest path problem on a graph with nonnegative edge-weights. Even with a data structure no more sophisticated than a binary heap, such a problem is solved in O(m log n) for arbitrary graphs, and O(n log n) for planar graphs, where n is the number of vertices (cells) and m is the number of edges. Hence for these planning methods in this domain, evaluating the plan-execution times is crucial to determining overall performance.

We thus first related the two planning methods. We then used tools from graph theory to analyze the travel distance of the robots for both planning methods in a unified framework and compared it to the optimal travel distance. We showed that the worst-case travel distance of GM and D* is at least on the order of (log |V| / log log |V|) |V| steps for graphs with |V| vertices, even if the graphs are planar. It had previously been reported that the travel distance of D* is good in typical robot domains, but it was unknown whether this robot navigation method is worst-case optimal, taking into account the lack of initial knowledge about the terrain. Our results establish that its worst-case travel distance is not optimal. However, it is not very badly sub-optimal either. The worst-case optimal travel distance is on the order of |V| while that of GM and D* is at most on the order of |V|^(3/2) steps, which provides a first step towards explaining the good empirical results that have been reported about GM and D* in the experimental literature. More generally, our results show how to use tools from graph theory to analyze the plan quality of practical planning methods for nondeterministic domains.

LOCALIZATION:

The localization problem is to determine the location of a robot, given a map of its environment. Solving the problem generally requires a reconnaissance plan in which the robot travels to gather more sensory data. We studied an idealized version of the problem, in which there is no actuator or sensor uncertainty. We studied localization on grids and in continuous environments, and with short-range (tactile) and long-range sensors. Our results turn out to be similar for the different models.

Localization is often studied in the context of an online analysis. The idea of an online analysis is to compare the distance traveled by the robot to the distance that would be traveled by an omniscient robot that knew its location at the outset, and sought only to verify that location. Our work takes a different tack. We believe that performance measures other than regret are of at least equal concern in real robot navigation. In particular, we utilized two measures: the worst-case performance ratio and the worst-case performance, where the travel distance is used as a measure of performance in both cases. We also performed an empirical assessment on different types of random gridworlds.

Dudek et al. proved that it is NP-hard to minimize the worst-case performance for robots with long-range sensors in polygonal regions. Their proof works essentially without change for robots with short-range sensors on grids. We proved a stronger result for the different models, showing that it is already NP-hard to find a plan whose worst-case travel distance is within a logarithmic factor of minimum. Thus, plans with near-optimal worst-case performance ratios cannot be found in polynomial time and it is reasonable to attempt to find plans with as good a performance ratio as possible.

In this context, we studied the performance ratio of a simple depth-first search method and of greedy localization, a localization method that moves the robot from its current location to the nearest informative location. Greedy localization is interesting because it has been proposed and employed in simulated implementations by several researchers. We found that both algorithms have very poor worst-case performance ratios, namely Omega(n) for long range sensors in polygons, and Omega(n/log n) for short range sensors in graphs. On the other hand, the worst-case performance of greedy localization is very good, namely between Omega(n log n/log log n) and O(n log n) for both sensor types.

Our results increase the understanding of the properties of popular localization methods. However, they also have another use because gridworlds are popular testbeds for planning with incomplete information and our results help one to understand whether they are indeed good testbeds. We found empirically that greedy planning methods that interleave planning and plan execution can localize robots very quickly on random gridworlds or mazes. Thus, they may not provide adequately challenging testbeds. On the other hand, we showed that finding localization plans that are within a log factor of minimum is NP-hard. Thus there are instances of gridworlds on which all greedy planning methods perform very poorly, and we showed how to construct them. These theoretical results help empirical artificial intelligence planning researchers to select appropriate planning methods for planning with incomplete information as well as testbeds to demonstrate them.

OTHER:

We have also started to develop the software and hardware infrastructure that will allow us to test some of the methods empirically both in simulation and on real robots. For example, we extended Teambots, an existing realistic robot simulation and navigation environment, with classes to simulate an ATRV robot. We then implemented algorithms for acquiring and maintaining maps of the terrain and for efficiently recomputing shortest paths to given goal coordinates when the robot discovers additional obstacles, allowing it to find shortest paths in totally or partially unknown or dynamic environments. This allowed us to implement both D* for goal-directed navigation in unknown terrain and

greedy mapping for mapping of unknown terrain, and to study their behavior empirically.

Our findings on on-line planning for teams of agents were presented in tutorials at AAAI 2006 and AAMAS 2006.

Findings: (See PDF version submitted by PI at the end of the report)

Findings: Overview

We found that in general, greedy search methods that re-plan after each step are computationally practical, perform very well empirically, and have quite good worst-case travel distance guarantees. For example, the Mars Rover Prototype search heuristic has only slightly super-linear worst-case distance. Most of the guarantees we found are extremely close to tight, within a log log factor; a few are tight to within a log n log log n factor. There does not appear to be a single analysis that applies to the three prototypical robot search tasks of mapping, reaching a goal, and determining one's location. Nonetheless, we found it fruitful to concentrate both design and analysis on aspects of the set of information that has been obtained. Thinking in these terms led to some unified analysis, and often led to new effective variants, of well-used methods.

We developed improved versions of D*, A*, and other classical search methods applied to robot navigation. Improvement was assessed empirically according to plan quality, and/or computational effort, rather than theoretical analysis.

We experimented with trail-laying robots, inspired by ant colony foraging. Although we found it feasible to equip robots with trail laying and sensing capabilities, and to navigate them, we did not find that ant-inspired search has the high quality travel distance guarantees that greedy methods tend to have. The trails that have been laid in the terrain do not seem to carry enough information. Both the greedy and the trail-laying methods are local search methods, in the sense that at each step the robot moves to a location near to its current location, and then re-plans. Why do the latter methods lack the high quality travel guarantees of the former? The difference appears to be that at each step, the trail-laying method only uses spatially local (though not necessarily temporally local) information, but the greedy methods use all information previously acquired.

The greedy and other known heuristics for the location or 'kidnapped robot' problem, have very poor worst-case performance ratios. We made a theoretical breakthrough on this problem, finding a near-tight approximation algorithm. The key idea is best explained in terms of the sets of information gathered. Previous algorithms had made search plans within the vicinity of the robot where the information previously gathered guaranteed knowledge of the terrain. Thus search was restricted to fully-known terrain. Our search strategy plans to halve the uncertainty within the larger portion of terrain that is, in a sense, at least half-known. The strategy acquires information either by following the plan successfully, or by acquiring information that disrupts the plan. In the latter case, what appeared to be the most likely outcome has been shown false. In either case, therefore, the amount of uncertainty is halved.

In the findings report following we describe several of our findings in some detail, and situate these in context of the area of search. For the full body of findings, and details of methods, experiments, and analysis, the reader is referred to the publications listed in the Products section of the final report.

Training and Development:

Our project is an interdisciplinary project that combines ideas from artificial intelligence, robotics, and theoretical computer science. One of the principal investigators is a researcher in artificial intelligence and robotics, the other principal investigator is a researcher in theoretical

computer science and operations research.

As we had hoped, our research topic has promoted exchange between the artificial intelligence and theory groups in the College of Computing, both in terms of research and training of students. For example, students from artificial intelligence, robotics, theoretical computer science, and mathematics have been part of our research group and worked on this project. Clearly, the willingness and ability of students to acquire knowledge from a variety of disciplines is more important than ever. Our project provides the students with insights into artificial intelligence and theoretical computer science as well as applications to robotics. This helps the students to understand that different disciplines have studied how to make good decisions, and that it is possible to combine ideas from different disciplines. It allows them to experience and overcome the problem that different disciplines have different terminology, assumptions, and approaches and enlarges the set of methods that they have available to solve decision-making problems and analyze the resulting solutions. To be more quantitative, of the 6 graduate students who have worked on this project, at least
5 did interdisciplinary research between their field and another area involving our project (e.g. theoretical computer science and robotics, mathematics and robotics, operations research and artificial intelligence). At last year's INFORMS Computing conference, members of our team gave three talks on the interface
between AI and OR.

One of the students who worked on our project is now a graduate student in computer science at Carnegie Mellon University. Our project also attracted a visitor from Sweden. Two other students who worked on the project completed the Ph.D. at Georgia Tech; one is now on the research staff in computer science at the University of Texas at Austin, working on DARPA robotics and planning projects, and the other is an assistant professor at the University of Wisconsin Oshkosh. Two other students expect to complete the Ph.D. within a year.

Outreach Activities:

Because the project is inherently interdisciplinary, it can be unclear which boundary crossings comprise our outreach activities. Throughout the project we have striven to reach the robotics, the artificial intelligence, and the theory communities. Our findings have been presented at high level conferences in all three fields, such as IROS, AAAI, SODA, etc. (see products). They have also been featured in conference plenary presentations and in well-attended tutorials, national and international. These outreach activities will described further in another section. The bulk of this section describes other outreach activities.

Sven Koenig was a special awards judge at the Intel International Science and Engineering Fair (ISEF) in both 2002 and 2004, representing the American Association for Artificial Intelligence (AAAI). Founded in 1950, ISEF is the world's largest pre-college science fair. It brings together over 1,200 students from 40 nations to compete for scholarships, tuition grants, internships, and scientific field trips.

Sven Koenig co-chaired the Student Abstract and Poster Program of the National Conference on Artificial Intelligence in 2002 (for the third time in a row). He was also mentor or panel member or mentor at the 2002 SIGART/AAAI Doctoral Consortium, the 2003 ICAPS Doctoral Consortium, the 2003 SIGART/AAAI/IJCAI Doctoral Consortium, the 2004 ICAPS Doctoral Consortium, and the 2004 AAMAS Doctoral Consortium. These events help graduate students of artificial intelligence by giving them feedback on their research and bringing them in contact with other junior and senior researchers in their area of research.

Sven Koenig also co-chaired the 3-day Symposium on Abstraction, Reformulation, and Approximation (SARA) in 2002. It has been recognized since the inception of Artificial Intelligence that abstractions, problem reformulations and approximations (AR&A) are central to human common-sense reasoning and problem solving and to the ability of systems to reason effectively in complex domains. AR&A

techniques have been used to solve a variety of tasks, including search and planning. The primary purpose of AR&A techniques in such settings is to overcome computational intractability. The SARA series is the continuation of two separate threads of workshops: AAAI workshops in 1990 and 1992, and an ad hoc series beginning with the 'Knowledge Compilation' workshop in 1986 and the 'Change of Representation and Inductive Bias' workshop in 1988 with followup workshops in 1990 and 1992. The two workshop series merged in 1994 to form the first SARA. Subsequent SARAs were held in 1995, 1998, and 2000. SARA 2002 was the most successful SARA yet. 51 researchers attended from countries around the globe, and 20 of the attendees were Ph.D. students.

Finally, Sven Koenig was on the organizing committee of the 2003 Americas School on Agents and Multiagent Systems. The purpose of this summer school was to educate graduate students on the fundamentals in the field of agents and multiagent systems. As explained on its website, there has been an exponential growth in the field with several major conferences, journals, international competitions and other significant events. Major new theoretical foundations are emerging, and there is an exciting growth in the practical applications. As new generations of graduate students immerse themselves in this field, it is critical to educate them in the fundamentals of this rapidly growing area. The 2003 Americas School on Agents and Multiagent Systems is the successor to the highly successful First Americas School on Agents and Multiagent Systems, which was held January 6-11, 2002 at the University of Southern California. The first school was filled to capacity, hosting 62 students from 19 universities, including five overseas universities.

Journal Publications

C. Tovey and S. Koenig, "Gridworlds as Testbeds for Planning with Incomplete Information", Proceedings of the National Conference on Artificial Intelligence, p. 819, vol. 1, (2000). Published

S. Koenig, "Agent-Centered Search", Artificial Intelligence Magazine, p. 109, vol. 22 / 4, (2001). Published

S. Koenig, "Minimax Real-Time Heuristic Search", Artificial Intelligence, p. 165, vol. 129, (2001). Published

S. Koenig and M. Likhachev, "Incremental A*", Advances in Neural Information Processing Systems, p. 1, vol. 14, (2001). Published

S. Koenig, C. Tovey and W. Halliburton, "Greedy Mapping of Terrain", Proceedings of the International Conference on Robotics and Automation, p. 3594, vol. 1, (2001). Published

C. Tovey and S. Koenig, "Greedy Localization", Proceedings of the International Conference on Intelligent Robots and Systems, p. 427, vol. 1, (2001). Published

S. Koenig and R. Holte (editors), "Symposium on Abstraction, Reformulation, and Approximation - Editorial", Lecture Notes in Artificial Intelligence, p. V, vol. 2371, (2002). Published

S. Koenig, Y. Smirnov, and C. Tovey, "Performance Bounds for Planning in Unknown Terrain", Artificial Intelligence Journal, p. 253, vol. 147, (2003). Published

J. Svennebring and S. Koenig, "Towards Building Terrain-Covering Ant Robots", Lecture Notes in Computer Science, p. 202, vol. 2463, (2002). Published

J. Svennebring and S. Koenig, "Trail-Laying Robots for Robust Terrain Coverage", Proceedings of the IEEE International Conference on Robotics and Automation, p. 75, vol. , (2003). Published

C. Tovey, S. Greenberg, and S. Koenig, "Improved Analysis of D*", Proceedings of the IEEE International Conference on Robotics and Automation, p. 3371, vol. , (2003). Published

C. Tovey, S. Koenig, "Minimizing Localization Travel Distance", IEEE Transactions on Robotics, p. , vol. , ( ). In revision

J. Svennebring and S. Koenig, "Building Terrain-Covering Ant Robots", Autonomous Robots, p. 313, vol. 16, (2004). Published

S. Koenig and Y. Liu, "Terrain Coverage with Ant Robots: A Simulation Study", Proceedings of the Fifth Autonomous Agents (Agents), p. 600, vol. , (2001). Published

A. Mudgal, C. Tovey, and S. Koenig, "AI&M 20-2004 Analysis of Greedy Robot-Navigation Methods", AI&M 1-2004. Eighth International Symposium on Artificial Intelligence and Mathematics, January 4-6, 2004, Fort Lauderdale, Florida., p. 173, vol. , (2004). Published

A. Ranganathan and S. Koenig, "PDRRTs: Integrating Graph-Based and Cell-Based Planning", Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS), p. , vol. , (2004). Accepted

C. Tovey and S. Koenig, "Improved Analysis of Greedy Mapping", Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS), p. , vol. , (2003). Accepted

Apurva Mudgal, Craig Tovey, Sam Greenberg, Sven Koenig, "Bounds on the Travel Cost of a Mars Rover Prototype Search Heuristic", SIAM Journal on Discrete Mathematics, p. 431, vol. 19, (2005). Published

Koenig, Tovey, Lagoudakis, Markakis, Kempe,Keskinocak,Kleywegt,Meyerson,Jain, "The Power of Sequential Single-Item Auctions for Agent Coordination
(Nectar Paper)", Proceeedings of the National Conference on Artificial Intelligence (AAAI), p. , vol. , (2006). Accepted

C. Tovey, M. Lagoudakis, S. Jain and S. Koenig, "The Generation of Bidding Rules for Auction-Based Robot Coordination", Multi-Robot Systems: From Swarms to Intelligent Automata, L. Parker, F. Schneider and A. Schultz (editors), p. 3, vol. 3, (2005). Published

D. Furcy, S. Koenig, "Limited Discrepancy Beam Search", Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), p. 125, vol. , (2005). Published

S. Koenig and M. Likhachev, "Fast Replanning for Navigation in Unknown Terrain", Transactions on Robotics, p. 354, vol. 21, (2005). Published

Yaxin Liu, Sven Koenig, "Existence and Finiteness Conditions for Risk-Sensitive Planning: Results and Conjectures", Proceedings of the International Conference on Uncertainty in Artificial Intelligence (UAI-05), p. , vol. , (2005). Accepted

M. Likhachev and S. Koenig, "A Generalized Framework for Lifelong Planning A*", Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), p. 99, vol. , (2005). Published

David Furcy and Sven Koenig, "Scaling up WA* with Commitment and Diversity [Poster Abstract]", Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), p. 1521, vol. , (2005). Published

Koenig, Likhachev, Liu, and Furcy, "Incremental Heuristic Search in Artificial Intelligence", Artificial Intelligence Magazine, p. 99, vol. 25, (2004). Published

S. Koenig, M. Likhachev, and D. Furcy, "Lifelong Planning A*", Artificial Intelligence Journal, p. 93, vol. 155, (2004). Published

Sven Koenig, Apurva Mudgal, Craig Tovey, "A near-tight approximation lower bound and algorithm for the kidnapped robot problem", Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithms (SODA 17), p. 133, vol. , (2006). Published

Yaxin Liu, Sven Koenig, "Risk-Sensitive Planning with One-Switch Utility Functions: Value Iteration.", Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-05), p. 987, vol. , (2005). Published

Books or Other One-time Publications

Web/Internet Site

URL(s):
http://www.cc.gatech.edu/fac/Sven.Koenig/greedonline
Description:
We put together the following webpages about greedy on-line planning, that contain results from this project and other projects:

http://www.cc.gatech.edu/fac/Sven.Koenig/greedonline
http://www.cc.gatech.edu/fac/Sven.Koenig/ants.html

We put our results about greedy on-line multi-robot coordination on the web-page:

http://www.idm-lab.org/project-c.html

Other Specific Products

Contributions

Contributions within Discipline:
It does not happen often that researchers from artificial intelligence, robotics, and theoretical computer science collaborate. This interdisciplinary project therefore allows us to lay much more rigorous algorithmical foundations for on-line planning in artificial intelligence and robotics than what could be done by researchers in any one of these disciplines alone. We analyze on-line planning methods and develop new methods. We have concentrated on methods for mapping of unknown terrain, goal-directed navigation in unknown terrain, and localization. To ensure impact, we have concentrated on methods that have successfully been used on real robots, including Mars rover prototypes, UGV Demo II vehicles, and robots for urban reconnaissance.

Overall, our research has resulted in a substantially better understanding of robot navigation in unknown terrain. Robot navigation in unknown terrain has been studied in both theoretical robotics and theoretical computer science. However, empirical robotics researchers have often developed their own planning methods. These planning methods have been demonstrated on mobile robots that solve complex real-world tasks and perform well in practice. However, they are very different from the

planning methods that have traditionally been studied in artificial intelligence, theoretical robotics, and theoretical computer science, and their properties are not yet well understood. We study the properties of these robot navigation methods (such as Planning with the Freespace Assumption, Greedy Mapping, Avoiding the Past, and Greedy Localization) formally with the goal to understand how good they really are and in which situations they should be used. The results provide guidance for empirical robotics researchers when to use which method and how to improve on them. We also study localization. Even though this is a central problem for mobile robots, the question of how difficult it is to localize a robot has not been studied in depth. Again, our results provide guidance to empirical robotics researchers for how to localize robots quickly by interleaving planning and plan execution.

Our research has resulted in new insights into the behavior of these on-line planning methods. For example, we analyzed the behavior of D* (probably the most popular method among robot practitioners) for goal-directed navigation in unknown terrain for the first time. Our research has also extended existing insights. For example, we showed that it is not only NP-hard to find a localization plan whose worst-case travel distance is minimal but that it is already NP-hard to find a localization plan whose worst-case travel distance is within a logarithmic factor of minimum. The results help robot practitioners to select appropriate methods for their problems, to understand the behavior of these methods better, and to improve them. Finally, our research has resulted in alternative implementation approaches for robots that cover terrain repeatedly even if some robots fail or they are moved without realizing this.

We reported first research results,
including papers at the top artificial intelligence and robotics conferences in the first year of our research. This trend continued in the second and third year, and extended to include theory conferences and journals as well.

One such contribution is the solution of the kidnapped robot approximability problem, including a novel 'half-safe' planning zone strategy. Another contribution is the establishment of theoretical underpinnings of risk-sensitive planning, and the development of computational practical methods for both risk-prone and risk-averse exponential utilities. Another contribution, to which this project provided a portion, is the development of an effective, generic auction-based heuristics agent coordination method.

Contributions to Other Disciplines:

Other disciplines to which the project has contributed are:

Biomimetics: development and analysis of ant-inspired trail-laying search.

Applied probability: development of necessary and/or sufficient conditions for existence of, and characterization of, optimal strategies for Markov decision processes with classes of nonlinear utility functions.

Contributions to Human Resource Development:

Two Ph.D. students graduated (Furcy, Liu). Training of four other Ph.D. students, an undergraduate, and a postdoc.

Contributions to Resources for Research and Education:

Craig Tovey has incorporated portions of this research into a core

Ph.D. course for both the Algorithms, Combinatorics, and Optimization program and the Industrial and Systems Engineering program at Georgia Institute of Technology. He has also incorporated portions of this research into invited talks at UC Berkeley and N.U. Singapore. Sven Koenig has given several tutorials that included portions of this research at ICRA 2002, AAAI 2002, AIPS 2002, ICRA 2003, and IJCAI 2003. This includes both tutorials on fast greedy search, given together with Anthony Stentz (Carnegie Mellon University, USA) and tutorials on ant robotics, given together with Israel Wagner (Technion and IBM Haifa Research Lab, Israel), Andrew Russell (Monash University, Australia), David Payton (HRL Laboratories, USA), and Richard Vaughan (HRL Laboratories, USA). Some of the research results have also been made available to a larger audience by publishing them on Sven Koenig's web pages and have been linked by web pages that provide educational resources to the general public, including the 'Artificial Intelligence Topics' web pages of the American Association of Artificial Intelligence.

Results from this research have been presented (by project participants and paper co-authors) at plenary presentations at the 2005 RoSS and 2006 AAMAS conferences. In addition, results have been disseminated in tutorials at AAMAS-06, AAAI-06, and ICRA 2006.

Contributions Beyond Science and Engineering:

Several of the results have the potential impact of being incorporated into, or inspiring, commercial, space, and homeland security technology. The most promising are the insights and improvements to agent-centered and goal-directed search, the Mars Rover prototype analysis, and the key idea in the new localization strategy of planning movement which is informative whether it succeeds or fails. Two other results that show promise, and to which the project has contributed in part, are the computationally practical incorporation of risk-averse and risk-prone utilities in planning, which should be applied to rescue, defense, and other high cost or payoff decisions, and the generic multi-agent auction-based coordination scheme.

However, it is too early to point to any specific areospace or defense application, or to a commercial product.

Categories for which nothing is reported:

Any Book

Any Product

Trail-Laying Robots for Robust Terrain Coverage

Jonas Svennebring
College of Computing
Georgia Institute of Technology
Atlanta, GA 30312-0280
jonas@cc.gatech.edu

Sven Koenig
College of Computing
Georgia Institute of Technology
Atlanta, GA 30312-0280
skoenig@cc.gatech.edu

Abstract— **Robotics researchers have studied robots that can follow the trails laid by other robots. We, on the other hand, study robots that leave trails in the terrain to cover closed terrain once or repeatedly. How to design such ant robots has so far been studied only theoretically for gross robot simplifications. In this paper, we describe for the first time how to build physical ant robots that cover terrain. We show that a modified version of node counting can model the behavior of the ant robots and report on first experiments that we performed to understand their behavior better. These experiments confirm that our ant robots indeed cover terrain robustly even if the trails are of uneven quality, the ant robots are moved without realizing this, or some trails are destroyed. Finally, we report the results of a large-scale experiment where ten simulated ant robots covered a factory floor of 25 by 25 meters repeatedly over 85 hours without any ant robots getting stuck.**

I. Introduction

How to cover terrain once or repeatedly is an important problem in mobile robotics, for example, in the context of mine sweeping, surveillance, surface inspection, and guarding terrain. Consequently, researchers have developed many coverage methods. Most of these methods assume that the robots know their location. The currently popular POMDP-based robot architectures [5] attempt to overcome this problem by providing robots with the best possible location estimates [13]. However, this approach is complicated and can be brittle for robots that are small and cheap and thus have extremely noisy actuators and sensors. In this paper, we therefore explore trail-laying robots (ant robots) as an alternative to this approach. Our inspiration came from those researchers who have studied ant robots that can follow the trails laid by other ant robots, similar to ants that lay and follow pheromone trails [1]. Ant robots that follow trails arrive at their destination without having to know their exact location, which eliminates solving difficult and time-consuming localization tasks. They need only simple sensors, namely sensors that are able to sense the trails, which are artificial landmarks that can be carefully designed to simplify sensing. We utilize a similar idea to build ant robots that cover closed terrain once or repeatedly without knowing where they are in the terrain. As before, they only have to leave trails in the terrain and sense the trails in their neighborhood. Different from before, however, they need to move away from the trails rather then follow them.

In previous work, we and other researchers have studied theoretically how to build ant robots for gross robot simplifications. Unfortunately, the resulting approaches are not very practical for implementations on physical ant robots. In this paper, we describe for the first time how to build physical ant robots that cover closed terrain once or repeatedly. Our ant robots robustly cover terrain even if they do not have any memory, do not know the terrain, cannot maintain maps of the terrain, nor plan complete paths. In particular, they cover terrain even if the trails are of uneven quality, some ant robots are moved without realizing this (say, by people running into them and pushing them accidentally to a different location) or some trails are destroyed.

We first discuss related work, the robot that we use and how we augmented it with trail-laying and trail-sensing hardware and ant-coverage software, and first experiments that we performed to understand its behavior better. We then show how a modified version of node counting can model its behavior. Finally, we report the results of a large-scale simulation experiment where ten ant robots covered a factory floor of 25 by 25 meters repeatedly over 85 hours without any ant robots getting stuck.

II. Related Work

Empirical researchers have studied physical ant robots that follow trails. Some researchers have imitated nature closely [8] while others only got inspiration from it. The ant robots have typically left short-lasting trails in the terrain, such as heat trails, alcohol trails, and odor trails [11], [12]. Some ant robots have also used virtual trails only [3], [9], [14]. We, on the other hand, study ant robots that leave actual trails in the terrain to cover it once or repeatedly. The different task demands both longer-lasting trails (to be able to mark terrain that has already been covered) and different ant-coverage software. There was an earlier effort by Gabrieli, Katan and Rogel at the Technion to build terrain covering ant robots that lay trails using an evaporating liquid but no results have been reported on the success of this project.

III. Theoretical Foundation

Theoretical researchers have studied ant robots that cover terrain for gross robot simplifications [6], [16]. For

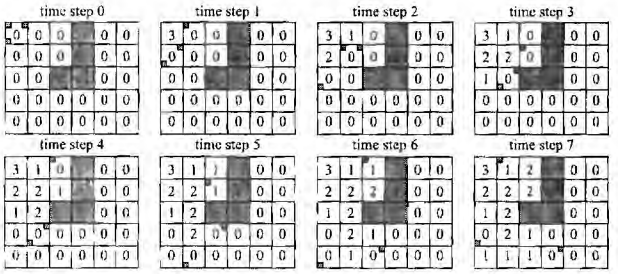

Fig. 1. Node Counting.

Fig. 2. Pebbles (left: birds-eye view; right: bottom view).

example, we have studied real-time search methods [7] for this purpose in earlier work. Real-time search methods are able to cover graphs repeatedly with a small cover time [15], [6], which suggests that they can be used to build ant robots that cover terrain once or repeatedly [6], [16]. The simplest real-time search method is probably node counting [10]. Ant robots that use node counting work as follows: Imagine that they operate on a four-connected grid and can move to each of the four neighboring cells of their current cell provided that the destination cell is traversable. They associate a number with each cell that corresponds to how often the cell has been visited by them. These numbers can be interpreted as markings that they leave at the cells. When an ant robot enters a cell, it increases the number of the cell by one. It then moves to the neighboring cell with the smallest number, breaking ties randomly. Figure 1 demonstrates the behavior of three ant robots that all use node counting. White cells are empty and gray cells are blocked. If a cell contains an ant robot, one of its corners is marked. Different corners represent different ant robots. For simplicity, we make the (unrealistic) assumption in the figure that the ant robots move in a given sequential order and that several ant robots can be in the same cell at the same time.

Single ant robots or teams of ant robots that use node counting cover grids and, more generally, all strongly connected graphs once or repeatedly. The ant robots do not even need to communicate with each other except via the markings. They only need to have very limited sensing and computational capabilities and do not need to know or learn a map of the terrain. They only have to leave markings in the terrain, sense markings at their neighboring

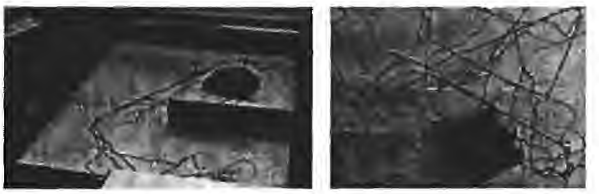

Fig. 3. Test Terrain (left: birds-eye view; right: top view).

cells, and change the marking of their current cell. Furthermore, ant robots that use node counting have advantages over ant robots that use other methods to cover terrain. For example, they navigate far more systematically than ant robots that perform random walks and, different from ant robots that use chronological backtracking (depth-first search), can be suspended and restarted somewhere else, without even knowing that they were moved or where they got restarted. This is important because sometimes ant robots might get pushed accidentally. Most of the time they will not even realize this. Ant robots that use node counting handle these situations automatically. It is known that the cover time of ant robots that use node counting can be exponential in (the square root of) the number of vertices on strongly connected undirected graphs [6], where the cover time is the time it takes to visit each vertex at least once. It is currently unknown whether the cover time is smaller on grids but experimental results indicate that it increases only linearly with the number of cells on grids and, for grids of a given size, remains small over time as the terrain is covered repeatedly. Unfortunately, despite these advantages, it is very difficult to build physical ant robots based on node counting and other real-time search methods. Their unrealistic assumptions include that the ant robots only move in discrete steps, that the ant robots have markings of a large number of different intensities available, that they can mark cells uniformly, and so on. Methods that cover continuous terrain [17] assume no actuator or sensor noise or largely depend on random motion. In this paper, we describe for the first time how to build physical ant robots that cover closed terrain once or repeatedly and demonstrate that they indeed cover terrain robustly even if the trails are of uneven quality, the ant robots are moved without realizing this, or some trails are destroyed. Figure 2 (left) shows our prototype ant robot, Pebbles, and Figure 3 (left) shows one of our test environments.

IV. The Ant-Coverage Hardware

For our research, we use a Pebbles III robot from IS Robotics, shown in Figure 2 (left) from birds-eye view and Figure 2 (right) from below. Its size with the tracks is 45 (width) by 44 (length) centimeters, and the size of its main body is 26 by 41 centimeters. Its on-board computer is based on a Motorola 68000 micro-controller

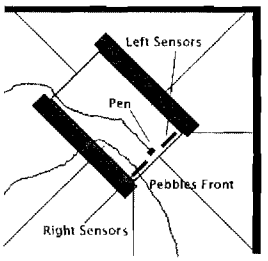

Fig. 4. Navigation Example.

that executes a special version of Common Lisp. It uses several peripheral micro-controllers to control its sensors and actuators, including its sonar sensors, its six infrared proximeters (sensing obstacles in the front, front-left, front-right, left, right, and rear of Pebbles), and its bump sensors for obstacle avoidance as well as its two motors for actuation. It navigates on two tracks and uses two re-chargeable 7.2V NiCd batteries that enable it to operate for about 30 minutes. We added both a trail-laying and a trail-sensing mechanism to Pebbles. The new components are marked in Figure 2 (right). Eventually, Pebbles will lay trails by dripping a fluorescence or phosphorescence substance. For the time being, however, it lays trails using a black pen with a thick tip that is mounted below the front center of its body (C). To detect trails, it uses two one-dimensional arrays of proximeters (trail sensors) that are mounted to the right (A) and left (B) of the pen, which allows it to sense its trails right away and avoids it getting trapped in local minima. Each trail-sensor array consists of four Sharp 2L01 proximeters and covers an area of about 4 by 1 centimeters. Each proximeter detects the amount of light that is reflected from an LED via the floor to it. This allows it to detect trails since there is less surface reflection in darker areas, that is, on trails. The signal from the proximeter is then sent through a multiplexer to an analog-to-digital converter integrated in an Atmel AVR Mega163 micro-controller (D) and sampled approximately 2000 times a second. The value that corresponds to the darkest area is stored until a read command is sent from the external computer via an RS232 interface (E) approximately every five times a second. The value is then thresholded, reported, and subsequently reset to zero. This allows Pebbles to move fast without missing trails.

V. The Ant-Coverage Software

The ant-coverage software on Pebbles implements a schema-based navigation strategy [2] with two behaviors that are active at the same time, namely an obstacle-avoidance behavior and a trail-avoidance behavior. Pebbles switches the schema-based navigation strategy off only in the rare occasion when one of its bump sensors triggers, that is, if it ran into an obstacle. In this case, it moves away from the obstacle for a short time before it resumes its normal operation. We use the example situation from Figure 4 to explain the obstacle-avoidance behavior, the trail-avoidance behavior, and their combination.

A. Obstacle-Avoidance Behavior

The obstacle-avoidance behavior moves Pebbles away from walls and is fairly standard. The obstacle-avoidance vector is the weighted sum of a number of vectors. There is one vector for each obstacle sensor. It starts at the obstacle sensor and points towards the center of Pebbles. The length of the vector is inversely proportional to the distance of the sensed obstacle from Pebbles. Its weight is proportional to the importance of the obstacle sensor. The weight of the front obstacle sensor is 1/10, the weight of the rear obstacle sensor is 1/20, the weight of the front left and front right obstacle sensors is 1/40, and the weight of the left and right obstacle sensors is 1/55. The weights reflect, for example, that obstacles in front of Pebbles are more important than obstacles in its rear. The lines in Figure 4 show the placement of the obstacle sensors and the distance from Pebbles to the closest wall. The front, front-left, left and rear obstacle sensors sense walls. The resulting obstacle-avoidance vector has length 0.1367 and points about 137 degrees to the right of Pebbles, suggesting to turn away from the walls to the left of it.

B. Trail-Avoidance Behavior

The trail-avoidance behavior moves Pebbles away from trails. Its vector points away from trails with a fixed length of 0.1. There are four proximeters in the left trail-sensor array and four proximeters in the right trail-sensor array. Each proximeter of the left (right) trail-sensor array that senses a trail changes the angle of the vector by 17 degrees to the right (left). Thus, if all eight proximeters sense trails, then the vector points straight ahead. If all four proximeters of the left trail-sensor array sense trails but no proximeter of the right trail-sensor array senses a trail, then the vector points 68 degrees to the right. A problem with this approach is that the trail-sensor array observes only a small area of the terrain. This makes it difficult for Pebbles to determine a good trail-avoidance vector based on the current information from the sensor array alone. Pebbles therefore calculates the direction of the new trail-avoidance vector as above but then adds the direction of the old trail-avoidance vector to it, weighted with a decay factor smaller than one. (If the resulting angle is larger than 90 degrees to the left or right, it gets reduced to 90 degrees.) This way, the new trail-avoidance vector is influenced not only by the current information from the trail-sensor array but also the information from the trail-sensor array in the recent past. If Pebbles continues to detect trails on the same side, it turns more and more

Fig. 5. Trail-Avoidance Behavior (left: bad; right: good).

sharply away from that side. If it stops detecting trails, it turns less and less sharply until it moves straight again. The decay factor ensures that the influence of trails decays over time, since they are further away from Pebbles and their actual location is no longer known with certainty. It cannot be set too low because otherwise Pebbles will make many small turns when it senses a trail. These turns need to be sharp to turn Pebbles sufficiently. Thus, the motion of Pebbles will be jerky when it senses a trail. On the other hand, the decay factor cannot be set too high either because otherwise Pebbles will continue to slowly turn even long after it has stopped sensing a trail. We determined experimentally that a decay factor of 0.5 resulted in smooth trajectories that turn Pebbles for only a short time after it senses a trail. So, if Pebbles continuously senses a trail on its left (that is, exactly one proximeter of its left trail-sensor array always senses a trail), then Pebbles will eventually turn 34 degrees to the right. Usually, the trail is no longer below its body before it achieves this turn angle, resulting in turns of only approximately 30 degrees before it moves straight again. The following table gives a fictitious example (that is different from the example from Figure 4) of exactly how the direction of the trail-avoidance vector is computed over time, assuming for simplicity that each trail is sensed by only one proximeter at a time. All angles are relative to Pebbles and were rounded to integers.

Time	Event	Old Angle (in degrees)	New Angle (in degrees)	
				0.0000
1	no trail	0.0000	$0.0000 - 0.5 \times 0.0000 =$	0.0000
2	trail on the right side	0.0000	$-17.0000 + 0.5 \times 0.0000 =$	-17.0000
3	no trail	-17.0000	$0.0000 - 0.5 \times 17.0000 =$	-8.5000
4	trail on the right side	-8.5000	$-17.0000 - 0.5 \times 8.5000 =$	-21.2500
5	trail on the left side	-21.2500	$17.0000 - 0.5 \times 21.2500 =$	6.3750
6	no trail	6.3750	$0.0000 - 0.5 \times 6.3750 =$	3.1875

For our main example, Figure 4 shows that the trail has been under the right trail-sensor array for the last 10 to 15 centimeters. The resulting vector of the trail-avoidance behavior has length 0.1 and points about 34 degrees to the left of Pebbles, thus suggesting to turn away from the sensed trail.

C. Combining the Behaviors

The obstacle-avoidance behavior and the trail-avoidance behavior both produce their own recommendation for how Pebbles should move. Pebbles always calculates the weighted average of the obstacle-avoidance and trail-avoidance vectors and moves in the direction of the resulting vector with a speed that is proportional to the length of this vector. The behavior of Pebbles is sensitive to the choice of these weights, and we thus optimized them by hand for the physical characteristics of Pebbles. The weight of the obstacle-avoidance behavior is larger than the weight of the trail-avoidance behavior since obstacle avoidance is more important than trail avoidance. The obstacle-avoidance behavior suggests to move right in the example from Figure 4, whereas the trail-avoidance behavior suggests to move left. This disagreement results in a short overall vector and thus a slow speed of Pebbles, which is desirable. The higher weight of the obstacle-avoidance vector results in an overall vector of length 0.0410 that points about 114 degrees to the right of Pebbles. Once Pebbles has turned away from the walls, its speed increases again and its navigation behavior is again mostly influenced by the trail-avoidance behavior.

The weight of the obstacle-avoidance behavior cannot be set too low because otherwise Pebbles can run into walls. On the other hand, it cannot be set too high either because otherwise Pebbles does not cover terrain close to walls and corners. Similarly, the weight of the trail-avoidance behavior cannot be set too low because otherwise Pebbles does not avoid previously covered terrain well enough and the cover time thus increases. On the other hand, it cannot be set too high either because otherwise trails can become barriers for Pebbles that are time consuming to cross and the cover time thus increases as well. To understand this phenomenon, assume that a trail separates two parts of a room that does not contain other trails. This trail is hard to cross for Pebbles because it gets repelled from it, as the last turn of Pebbles in Figure 5 (left) shows. Furthermore, the trail gets reinforced every time Pebbles approaches it but does not cross it. The emerging barrier can only be crossed easily once the trail density in the part of the room that Pebbles is in has become sufficiently high. Thus, the weight of the trail-avoidance behavior needs to get tuned carefully to allow Pebbles to cross orthogonal trails. Then, Pebbles first turns, say right, to turn away from the trail but continues to move forward and thus crosses the trail before turning. It then turns left, again to turn away from the trail, and eventually continues on its old trajectory, as shown in Figure 5 (right). If the weight of the trail-avoidance behavior is tuned carefully, this behavior clearly dominates, as shown in Figure 6.

VI. Experiments

We conducted several experiments to evaluate the performance of Pebbles. We used areas of sizes from 2 by 2.5 meters to 3 by 4.5 meters. Their floor was covered with white paper, and they were surrounded with brown cardboard walls.

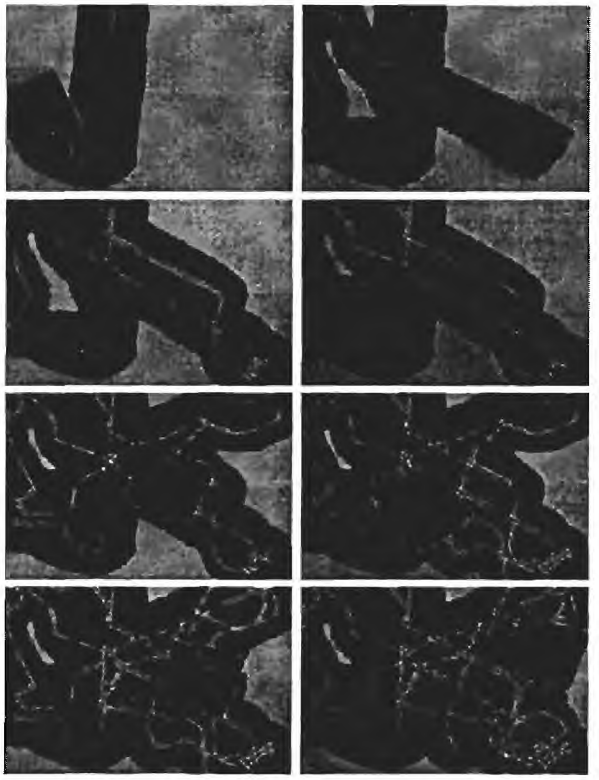

Fig. 6. Start of First Terrain Coverage (equal time steps).

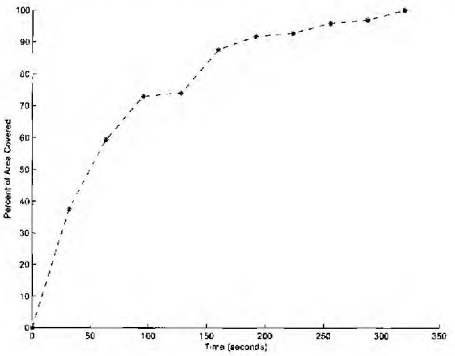

Fig. 7. Area Covered.

A. Regular Coverage

We first verified that Pebbles indeed covers terrain of different shapes multiple times without getting stuck, including the one shown in Figure 3 (left). Figure 6 shows the beginning of an example run in a simple rectangular terrain, where we shaded the covered area by hand in dark gray. We then measured the cover times for repeated terrain coverage. We say that the first coverage of the terrain is completed when each patch has been swept at least once by the body of Pebbles (except for patches close to walls). We say that an additional coverage of the terrain is completed when each patch has been swept at least once by the body Pebbles after the previous coverage was completed. For example, Pebbles covered an obstacle-free terrain of size 2 by 2.5 meters in 320 seconds. Figure 7 shows the covered area as a function of time, until the first coverage is completed. Pebbles covered terrain very quickly at the beginning (when it is easy to find uncovered areas since there are so many of them) but needed more time to find uncovered areas towards the end of the first coverage. This is true for the subsequent coverages as well. Pebbles needed 329 seconds for the second coverage and 489 seconds for the third coverage. The cover times increase because the terrain gets saturated with trails. This causes two problems. It makes it harder to place new trails and decreases the influence of each newly placed trail on the navigation behavior of Pebbles. Although more complex ant-coverage hardware and software will certainly be able to shorten the cover time, the current software of Pebbles does well given the small sensor field. For example, it faired well compared to a navigation behavior where Pebbles moves forward (without laying trails) while avoiding obstacles. In this case, Pebbles moved along the walls and covered about 50-60 percent of the terrain in 451 seconds but never covered the terrain completely in a reasonable amount of time.

B. Error Conditions during Coverage

One of the attractive properties of our ant-coverage software is that it covers closed terrain robustly even in situations where the trails are of uneven quality, where Pebbles is moved without realizing this, and where some trails are destroyed. We demonstrated the latter two properties earlier for rather unrealistic robot simulations [6] and demonstrate in the following that they continue to hold on Pebbles.

1) Trails of Uneven Quality: First, we measured the cover time when the pen of Pebbles was nearly exhausted. This is important because its pen is not constantly refilled with ink and its trails thus get lighter over time and harder to detect. This makes it more likely that Pebbles misses trails. Since Pebbles moves at different speeds, the pressure on the pen changes and the trails are not only faint but also of uneven quality. Since the intensity of the trails adds up over time, Pebbles continued to cover terrain robustly although the resulting cover time of 573 seconds is larger than the regular cover time of 320 seconds. Its coverage became also more uneven since some trails were stronger than others and became barriers for Pebbles.

2) Moving Pebbles: Second, we measured the cover time when Pebbles was moved without realizing this. This is important because people or other ant robots can easily run into Pebbles and accidentally push it to a different location. In the middle of a run, we moved Pebbles twice

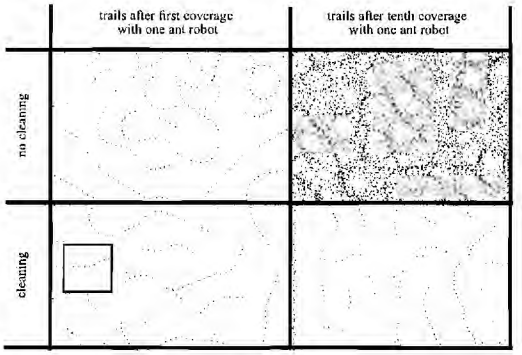

Fig. 8. Trails (with and without cleaning).

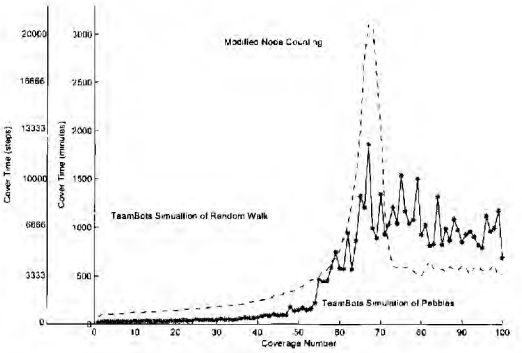

Fig. 9. Cover Time.

by a couple of meters in a random direction. Pebbles coped well with this problem since the ant-coverage software does not need to know its current location. The resulting cover time of 385 seconds is larger than the regular cover time of 320 seconds but the difference is not significant. This is not surprising since one of the random displacements moved Pebbles to the start corner (which increases the cover time) but the other one moved Pebbles to a location that it had not covered before (which decreases the cover time).

3) Removing Patches of Trails: Third, we measured the cover time when some trails were destroyed. This is important because trails can get destroyed accidentally due to wind, dust, rain, humans, or other ant robots. After Pebbles had covered about 90 percent of the terrain, we randomly placed three sheets of paper of size 15 by 20 centimeters on areas that it had already covered and that thus contained trails. Pebbles covered the three areas again only 103 seconds later since it was drawn to each area once it had sensed part of it and noticed that it did not contain trails.

VII. Computational Models of Ant Robots

So far, we have shown that our ant-coverage software makes Pebbles cover closed terrain robustly. We confirmed these results in TeamBots [4], a realistic robot simulator, with only slight modifications to the ant-coverage software. TeamBots has no battery limit and thus allowed us to extend Figure 7 to a large number of coverages. Note that the cover times of Pebbles cannot be compared to the cover times in simulation since the simulation uses its own clock, which does *not* correspond to actual time. The top row of Figure 8 shows that the terrain gets saturated with trails over time. (The square in the lower left corner corresponds to the simulated Pebbles.) The graph "TeamBots Simulation of Pebbles" in Figure 9 shows how this causes the cover time to increase for a terrain of size 10 by 10 meters. Eventually, the terrain is completely saturated with trails and the behavior of the simulated Pebbles degrades to a random walk, resulting in a cover time that is 40 times larger than the initial cover time. The cover time of node counting (not shown in the figure), on the other hand, remains small over time since node counting does not model that the terrain gets saturated with trails over time. We therefore modify node counting to provide a better computational model of Pebbles. Remember that ant robots that use node counting increase the number of their current cell by one and then move to the neighboring cell with the smallest number, breaking ties randomly. Ant robots that use the modified version of node counting, on the other hand, increase the number of their current cell by one only with probability $(k-x)/k$, where x is the current number of the cell and k is a constant (we use $k = 170$). Otherwise they leave the number of their current cell unchanged. Then they move to the neighboring cell with the smallest number, breaking ties randomly. To understand the idea behind the modified version of node counting, assume that each cell is divided into k small areas that are initially unmarked. Let x be the number of marked areas. If the ant robots randomly select one area of their current cell and mark it, then x increases by one with probability $(k-x)/k$, the probability that the chosen area was unmarked. Otherwise, x remains unchanged. The number of marked areas of each cell corresponds to its number. Thus, the probability with which ant robots that use the modified version of node counting increase the number of a given cell gets smaller and smaller over time, until the number of the cell is k and then does not change any longer. The modified version of node counting is a somewhat simplistic computational model of Pebbles but does model that it becomes harder and harder to add trails to areas that already contain a large number of trails, and that the terrain eventually gets saturated with trails and the behavior of Pebbles then degrades to a random walk. The modified version of node counting also models an interesting effect that we did not anticipate. Figure 9 shows that the cover times of both ant robots that use the modified version of node counting (in an obstacle-free grid of size 15 by 15 cells) and the

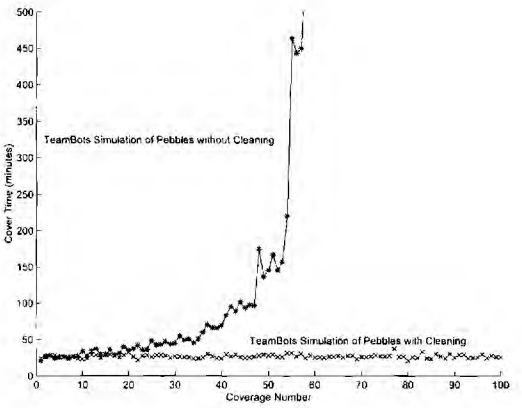

Fig. 10. Cover Time.

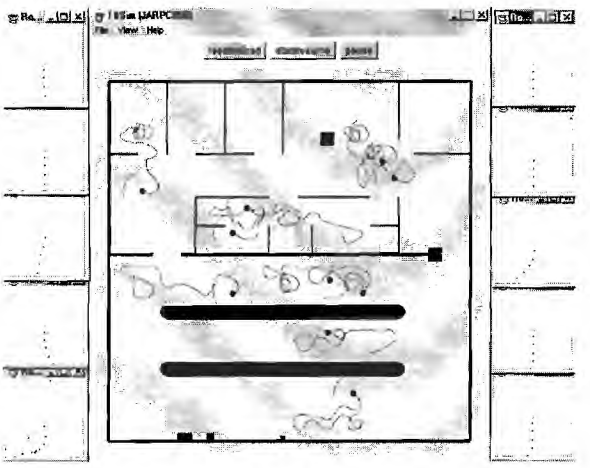

Fig. 11. Large-Scale Simulation Study.

simulated Pebbles (in an obstacle-free terrain of size 10 by 10 meters) peak and only then reduce to the cover time of a random walk, at which point the cover time does not change any longer. The modified version of node counting predicted the peak first and only then did we run experiments with our ant robots to verify it. We verified the cover time of a random walk by saturating the terrain with trails. It turns out that the peak is due to local minima in the trail density. For example, consider an area that is not completely saturated with trails but enclosed by areas that are completely saturated. It then takes both ant robots that use the modified version of node counting and the simulated Pebbles a long time to leave this area, longer than a random walk, since they first need to increase the trail density in their area to that of the surrounding ones, which takes time. This explains the peak. We assumed that each cell was divided into k small areas. This number is a parameter that determines how quickly the terrain gets saturated. We determined it empirically for our example to make the peaks of the cover times coincide.

VIII. Scaling Up

It is undesirable that the terrain gets saturated with trails over time since this increases the cover time. Thus, the trails need to either evaporate or get removed to keep the cover time small in the long run. Evaporating trails are problematic because the evaporation rate needs to get optimized for each application, for example, the size of the terrain. Thus, we propose to use long-lasting trails that Pebbles removes itself. However, the pen that Pebbles currently uses is not suited for this purpose. Instead, we simulated trails that consist of drops of a fluorescence or phosphorescence substance and used a cleaning method that removes all trails in two cleaning areas. Depending on the trail material, the trails could be removed with brushes, vacuum cleaners, heat (for alcohol trails), and light (for some photo chemicals) but it is future work for us to build such hardware. The bottom row of Figure 8 shows that this prevents the terrain from getting saturated with trails and Figure 10 shows that the cover time remains a small constant even after a large number of coverages in a terrain of size 10 by 10 meters. We also confirmed that the simulated Pebbles with the cleaning method continues to cover closed terrain robustly even in situations where it is moved without realizing this and where some trails are destroyed.

We then performed a large-scale simulation study to demonstrate that a large team of our ant robots with the cleaning method covers a large terrain repeatedly over long periods of time without ant robots getting stuck. We placed ten ant robots into an area of 25 by 25 meters that resembled a factory floor with two production lines and a number of office rooms, shown in Figure 11. This complex but very realistic environment is very difficult to cover due to its many narrow passages. We stopped the experiment after the ant robots had covered the factory floor for 85 hours without getting stuck. This result is important because teams of ant robots cover closed terrain faster and are more fault tolerant than single ant robots. The trails also coordinate the ant robots implicitly and allow them to cover terrain faster than without any communication.

IX. Conclusions

In this paper, we have described how to build physical ant robots that leave trails in the terrain to cover closed terrain once or repeatedly. The ant robots do not need to be localized, which completely eliminates solving difficult and time-consuming localization problems. We showed that a modified version of node counting can model the behavior of the ant robots and showed experimentally that physical ant robots robustly cover terrain even if the trails are of uneven quality, the ant robots are moved without realizing this (say, by people running into them), and some trails are destroyed. We also showed how ant

robots can keep the cover time small when repeatedly covering terrain, namely by removing old trails. A large team of simulated ant robots covered a large terrain repeatedly over long periods of time without any ant robots getting stuck. We are now working on demonstrating the advantages of teams of ant robots on physical ant robots. The purpose of this article was to demonstrate the robustness of our minimalistic ant robots despite their limited ant-coverage hardware and simplistic ant-coverage software. We are now working on ant-coverage software that decreases the cover time of our ant robots even more while continuing to let them cover closed terrain robustly without knowing where they are. We are also working on comparing our ant robots to other coverage algorithms, including more traditional ones.

ACKNOWLEDGMENTS

We thank Ashwin Ram for making his hardware available to us. The Intelligent Decision-Making Group is partly supported by NSF awards under contracts IIS-9984827, IIS-0098807, and ITR/AP-0113881 as well as an IBM faculty partnership award. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations and agencies or the U.S. government.

X. REFERENCES

[1] F. Adler and D. Gordon. Information collection and spread by networks of patrolling ants. *The American Naturalist*, 140(3):373–400, 1992.

[2] R. Arkin. *Behavior-Based Robotics*. MIT Press, 1998.

[3] T. Balch and R. Arkin. Avoiding the past: A simple, but effective strategy for reactive navigation. In *International Conference on Robotics and Automation*, pages 678–685, 1993.

[4] T. Balch and A. Ram. Integrating robotics research with JavaBots. In *Proceedings of the AAAI Spring Symposium*, 1998.

[5] S. Koenig and R.G. Simmons. Xavier: A robot navigation architecture based on partially observable Markov decision process models. In D. Kortenkamp, R. Bonasso, and R. Murphy, editors, *Artificial Intelligence Based Mobile Robotics: Case Studies of Successful Robot Systems*, pages 91–122. MIT Press, 1998.

[6] S. Koenig, B. Szymanski, and Y. Liu. Efficient and inefficient ant coverage methods. *Annals of Mathematics and Artificial Intelligence*, 31:41–76, 2001.

[7] R. Korf. Real-time heuristic search. *Artificial Intelligence*, 42(2-3):189–211, 1990.

[8] D. Lambrinos, R. Möller, R. Labhart, R. Pfeifer, and R. Wehner. A mobile robot employing insect strategies for navigation. *Robotics and Autonomous Systems*, 30:39–64, 2000.

[9] D. Payton, M. Daily, B. Hoff, M. Howard, and C. Lee. Autonomy-oriented computation in pheromone robotics. In *Proceedings of the Autonomous Agents Workshop on Autonomy Oriented Computation*, 2001.

[10] A. Pirzadeh and W. Snyder. A unified solution to coverage and search in explored and unexplored terrains using indirect control. In *Proceedings of the International Conference on Robotics and Automation*, pages 2113–2119, 1990.

[11] R. Russell. *Odour Sensing for Mobile Robots*. World Scientific, 1999.

[12] R. Sharpe and B. Webb. Simulated and situated models of chemical trail following in ants. In *Proceedings of the International Conference on Simulation of Adaptive Behavior*, pages 195–204, 1998.

[13] S. Thrun. Probabilistic algorithms in robotics. *Artificial Intelligence Magazine*, 21(4):93–109, 2000.

[14] R. Vaughan, K. Stoey, G. Sukhatme, and M. Mataric. Whistling in the dark: Cooperative trail following in uncertainty localization space. In *Proceedings of the International Conference on Autonomous Agents*, pages 187–194, 2000.

[15] I. Wagner, M. Lindenbaum, and A. Bruckstein. Efficiently searching a dynamic graph by a smell-oriented vertex process. *Annals of Mathematics and Artificial Intelligence*, 24:211–223, 1998.

[16] I. Wagner, M. Lindenbaum, and A. Bruckstein. Distributed covering by ant-robots using evaporating traces. *IEEE Transactions on Robotics and Automation*, 15(5):918–933, 1999.

[17] I. Wagner, M. Lindenbaum, and A. Bruckstein. MAC vs. PC: Determinism and randomness as complementary approaches to robotic exploration of continuous unknown domains. *International Journal of Robotics Research*, 19(1):12–31, 2000.

Understanding and Improving On-Line Methods for Robot Navigation with Incomplete Information

Sven Koenig

Craig Tovey

1 Introduction

A difference between search in robotics and typical search testbeds in artificial intelligence, like the eight puzzle or the Rubik's cube, is that search spaces in robotics are continuous and thus need to get modelled as discrete spaces. This discretization might not be simple because the space may contain complex obstacles of varying scale. In this report on the results of the NSF-funded research project, we therefore first discuss how to model such search spaces as graphs, which presumably could then be searched with any search method already discussed. However, once discretization is accomplished, a more fundamental difference between search in robotics and typical AI search testbeds presents: Robots typically have only incomplete information about the search space. They cannot determine in advance the observations their sensors will make after they have moved, nor the feasibility or effects of future moves. Complete AND-OR graph searches could in principle be used to find optimal plans but are often computationally intractable since the robots have to find prohibitively large conditional plans that solve the planning tasks. Yet, search has to be fast to move robots in real-time. Thus, one needs to speed up search by developing robot-navigation methods that sacrifice the optimality of the resulting robot paths. We will discuss and analyze two greedy on-line search techniques, namely agent-centered search and assumption-based planning, that make search under incomplete information fast, yet result in reasonably short paths. We draw examples principally from the class of grid graphs. These graphs are widely used to model space, lend themselves readily to illustration, and form a challenging testbed for search with incomplete information.

2 Search Spaces

If a robot is mobile, omnidirectional, and not subject to acceleration constraints, its configuration can be defined by its location in its workspace. In this case, the robot workspace is identical to its configuration, or *state* space.

In some other cases, one must distinguish between the robot workspace, and the robot configuration space. From a search planning point of view, the latter always is the search space.

Consider a motion-planning problem in a two-dimensional workspace with a robot arm that has two joints and is bolted to the ground at the first joint, as shown in Figure 2(a). An obstacle blocks one quadrant of the plane. The robot arm has to be moved from its start configuration to some goal configuration. This is a search problem whose states are given by the two joint angles. The corresponding configuration space is shown in Figure 2(b). The region in the figure wraps around because angles are measured mod 2π, so that the four corners depicted represent the same point. (One could also include the joint velocities to model constraints on both location and acceleration, in a 4-dimensional configuration space.)

Some points in the configuration space are impossible states because the robot arm would intersect an obstacle in the workspace. In other examples a point might be impossible because a joint angle is out of range. These impossible, or *blocked* states form obstacles in configuration space, obstacles that need not resemble the obstacles in the robot workspace, in size or shape. The possible, or *unblocked* configuration states form the *freespace*. To solve the problem, one finds a path in freespace from the start configuration point to the goal configuration point.

There will be different valid choices to define a configuration space. If the only objective of the planning is feasibility, that is, to find a path that does not traverse any blocked points, then the choice is usually unimportant. On the other hand, if the planning objective includes optimization of some cost, such as minimizing time or fuel, then it is preferable to define the configuration space so that this cost is represented as a norm-induced or other natural metric on the space. For example, horizontal movement in Figure 2(b) requires both joints to actuate. If only the first joint actuates, movement will occur along the diagonal with slope 1. Thus horizontal movement is more costly than diagonal movement, contrary to what the figure suggests visually. It may then be preferable to define the vertical axis of the configuration space as $\theta_2 - \theta_1$, as shown in Figure 2(c).

For simplicity, we employ the convention of saying that the robot is at a point, in configuration space, to mean that the configuration of the robot is at a point in configuration space. Thus, considering Figure 2, we say that the robot moves from the start point to the current point to the goal point, whereas it is actually the configuration of the robot arm that moves so.

Robot motion in workspace is, of physical necessity, continuous, but search in freespace is usually planned as search on a discrete structure, typically a graph. At some point, one must decide what the atomic actions are, that are available to the search planner. Therefore, one often needs to model a continuous freespace as a discrete space. For example, one level of a hierarchical control system may assure error-free sensing and discrete motion to the higher levels. In the following, we initially consider two-dimensional configuration spaces and describe two different ways of discretizing their freespace into graphs.

- One can discretize a 2D configuration space using rectangular cells or another regular tessellation of the plane, where a cell is considered unblocked if and only if it does not contain obstacles. Ordinarily one discretizes so that the robot can move between neighboring unblocked cells, that is, cells in freespace that share a border. One

Figure 1: Robot Arm Workspace and Two Configuration Spaces

problem is the granularity of the discretization. If the cells are large, then an existing path might not be found, as shown in Figure 3. The circle and cross show the start and goal point, respectively, and white and black areas are freespace and obstacles, respectively, in the continuous configuration space. On the other hand, the smaller one makes the cells, the more numerous they becomes, which results in a quadratic explosion. It can therefore make sense to have cells of different sizes: small cells close to obstacles in configuration space to be able to find gaps between them and large cells otherwise. One particular method for obtaining a non-uniform discretization is the parti-game algorithm. It starts with large cells and splits them as needed while trying to move the robot from the cell with the start point to the cell that contains the goal point (goal cell). We illustrate a simple version of the parti-game algorithm in Figure 4. It starts with a uniform coarse-grained discretization of the configuration space. The vertices of the graph represent the cells, and edges connect vertices that correspond to neighboring cells. Thus, it initially ignores obstacles (freespace assumption) and makes the optimistic assumption that it can move from any cell to each neighboring cell. It uses the graph to find a shortest path from its current cell to the goal cell. It then follows the path

Figure 2: Robot Arm Planning and Replanning

(a) Uniform Coarse-Grained Discretization

(c) Uniform Fine-Grained Discretization

Figure 3: Cell-Based Discretization

by always moving towards the center of the successor cell of its current cell. If it gets blocked by an obstacle (which can be determined in the workspace and thus without modeling the shape of obstacles in configuration space), then it is not always able to move from its current cell to the successor cell and therefore removes the corresponding directed edge from the graph. It then re-plans and finds another shortest path from its current cell to the goal cell. If it finds such a path, then it repeats the process. If it does not find such a path, then it uses the graph to determine from which cells it can reach the goal cell (solvable cells) and from which cells it cannot reach the goal cell (unsolvable cells, shaded grey in

Figure 4: Parti-Game Algorithm

the figure). It then splits all unsolvable cells that border solvable cells (and have not yet reached the resolution limit) along their shorter axis. (It also splits all solvable cells that border unsolvable cells along their shorter axis to prevent neighboring cells from having very different sizes, which is not necessary but makes it efficient to determine the neighbors of a cell with kd-trees.) It deletes the vertices of the split cells from the graph and adds one vertex for each new cell, again assuming that it can move from each new cell to any neighboring cell and vice versa. (It could remember where it got blocked by obstacles to delete some directed edges from the graph right away but does not. Rather, it deletes the edges automatically when it gets blocked again by the obstacles in the future.) It then re-plans and finds another shortest path from its current cell to the goal cell and repeats the process until the goal cell is reached or no cell can be split any further because the resolution limit is reached. The final discretization is then used as initial discretization for the next motion-planning problem in the same configuration space.

- One can also discretize freespace by picking a number of points in freespace (including the start and goal points) and connecting each pair of points with a shortest path (or the trajectory of some simple controller) provided that it is unblocked (which can perhaps be determined in the workspace and thus without modeling the shape of obstacles in configuration space). In case all obstacles are polygons, one can use the corners of the polygons as points (in addition to the start and goal points). The resulting graph is called the visibility graph. A shortest path on the visibility graph is also a shortest path in freespace. However, the obstacles are often not polygons and can only be approximated with complex polygons, resulting in a large number of points and thus large planning times. Probabilistic roadmaps (PRMs) therefore choose a number of points in freespace randomly (in addition to the start and goal points). The larger the number of points is, the larger the planning time tends to be. On the other hand, the larger the number of points is, the more likely it is that a path is found if one exists and the shorter the path tends to be. So far, we have assumed that the freespace is first discretized and then searched. However, it is often faster to pick random points during the search. Rapidly-exploring random trees (RRTs), for example, grow search trees by picking points randomly and then trying to connect them to the search tree via unblocked straight lines.

The first method generalizes naturally, but expensively, to higher dimensions. The second method generalizes also. Many methods, both deterministic and randomized, have already been developed to select the point set. Algorithms from computational geometry might lead to new practical methods with strong performance guarantees. These algorithms select an approximating set of representative points from the freespace, in the sense that the length of a shortest path on these representative points is with high probability within some small factor of the length of a shortest path in freespace. Discretizing a freespace in high dimension can be difficult because the obstacles in configuration space can have complex shapes even if the obstacles in the workspace have very simple shapes.

There are also other discretization methods, for example, using Voronoi graphs, which have the advantage that paths between obstacles remain as far away from the obstacles as possible and thus maximize the safety distance to them.

Once a configuration space has been discretized, it is only a slight further abstraction to model the freespace as a graph $G = (V, E)$. We have anticipated this abstraction in the description of the parti-game algorithm. Each vertex $v \in V$ of the graph represents a point in the discrete space (which may in turn represent a point or atomic region in configuration space). The robot's configuration, at any instant, is represented by exactly one vertex. By our convention, we simply say that the robot is at a vertex, although this simpler terminology is strictly true only if the configuration and work spaces are identical.

Each pair of configurations, v_1, v_2, that can be reached from the other by an atomic robotic movement is represented by an edge $(v_1, v_2) \in E$ in the graph. In the case of a 2D grid, the vertices of the graph represent the unblocked cells, and edges connect vertices that correspond to unblocked cells that share a border.

The graph model just defined has proven to be extremely useful for algorithm specification, implementation, and analysis. In some situations, one varies the model. If some movements, for example sliding down an unclimbable slope, are not reversible, some edges of the graph may be directed. If not all atomic movements have equal cost, the edges may be need to be labelled with weights. For example, if the granularity of the discretization is not uniform, differing costs may be required to move between neighboring cells of various sizes.

Regardless of what variant of graph model is used, the graph need not provide a complete description of the problem. There may be additional information, for instance about

sensor capabilities, that is not encapsulated in G. If all information were complete *a priori*, discretization ordinarily would reduce the search problem to a deterministic search problem of manageable size. This deterministic search problem could then be solved with any of several standard search methods. In this research project, we focus on robots that search without complete information about the space and/or their configuration.

3 Search with Incomplete Information

3.1 Graphical Base Model

Throughout this report, we work principally with one core set of assumptions, which we call the *base model*. As suggested by the colloquial meaning of the phrase, the base model is a set of minimum capabilities.

Base Model

Robot motion in freespace may be abstracted to movement and sensing on an undirected graph with the following properties:

1. The robot occupies one vertex at a time.
2. The robot moves deterministically by traversing an edge from one incident vertex to the other incident vertex, without error.
3. The robot's sensory, memory, and computational capabilities always provide without error, at minimum, the following information:
   (a) Observation of the vertex it occupies.
   (b) Observation of each edge incident on the vertex it occupies, including observation of the adjacent vertex.
   (c) Memory of all previous observations and edge traversals.
   (d) Recognition, upon observation, of any vertex or edge that it has previously observed.
4. Sensing is uniform over time. That is, the robot's sensors report the same information each time the robot occupies the same vertex.

By definition, the robot cannot leave the connected component in which it starts. Therefore we generally assume that the base model graph is connected.

Many scenarios of robot movement and sensing are consistent with the base model. We next describe four such scenarios. Consistency means that all base model capabilities are satisfied in the scenario, not vice-versa. Different robots in different environments may achieve these capabilities by different means. For example, GPS for a mobile robot would uniquely identify every vertex, thus providing recognition of vertices and edges previously observed. GPS in effect provides each vertex with a unique identifier that the robot observes. In an environment without GPS, a mobile robot without actuator uncertainty, could, with a compass, retain its entire history of movement. It could then always compute its current location relative to its starting point, thus providing recognition of any previously observed edge or vertex.

2D tactile gridworld: a mobile robot with compass and short-range sensing in a 2D regular gridworld. This scenario fits the base model as follows: each unblocked cell in the gridworld corresponds to a vertex in G; each boundary between two unblocked cells corresponds to an edge. Ordinarily a boundary must have nonzero length, so that a vertex may have up to four neighbors, one each in the directions N,S,E,W. The resulting grid graph is termed a 4-grid. Many of our illustrations will be drawn from this scenario. (If boundaries of zero length are permitted, a vertex may have up to eight neighbors, resulting in an 8-grid.) Let us verify that the base model capabilities hold: the robot occupies one cell at a time (1); it can move to an unblocked cell in any of the four compass directions without actuation error (2); the short range sensors determine, without error, in which of the four compass directions the adjacent cell is unblocked. Observation of a vertex amounts to perceiving that the vertex exists, that is, that the cell exists in freespace. As explained above, by retaining its history of movement, the robot can always determine its location relative to its starting point, and thus can recognize previously observed vertices and edges (3). Property 4 holds for this and other scenarios because the environment is static and sensing is deterministic.

k-jointed robot arm: A k-jointed robot arm's k-dimensional configuration space, discretized. The robot configuration is a vector in $[0, 2\pi]^k$ representing the set of joint angles. Since the mapping from the workspace to the configuration space is continuous, movement in the configuration space is continuous. Hence the discretization satisfies properties 1 and 2. The sensory requirements 3 amount to knowing the current joint angles, and detecting obstacles in the workspace that are close to the arm, close in the sense of being adjacent in the discretization. In this scenario, rectangles with different length sides are superior to squares, because a small change in the first joint angle has more potential effect in the workspace than a small change in the last joint. Therefore, the higher numbered the joint, the coarser grained should be its dimension.

Line-of-sight-sensor continuous polygonal model: A mobile robot moving continuously with long range sensors in a polygon, which is discretized by a representative point set. A polygon is a piecewise-linear simple closed curve in the plane. In the standard continuous polygonal model, the robot is a point which moves continuously within the polygon's interior $int(P)$. There may also be finitely many obstacle polygons $P_1 \ldots P_m$, whose exteriors $ext(P_j)$ must contain the robot location. An obstacle polygon may be degenerate, that is, it needs not be a closed curve. Instead it can be a barrier composed of line segments, in which case its exterior is defined to be its complement. Therefore, the robot may only move within the region $int(P) \cap \{\cap_{j=1}^{m} ext(P_j)\}$. The boundary of this freespace is composed of line segments. The robot's long range sensors can detect the distance to the nearest line segment in any direction.

A finite set of representative points in freespace forms the set V of vertices of the graph. Since any real robot occupies a nonzero volume, this discretization helps compensate for the idealization of the robot as a point. Any vertex w that is visible from vertex v, that is, such that the line segment $\overline{vw}$ does not intersect any line segment of the polygon (or obstacle polygons), is eligible to be connected to v by an edge in the graph. Often one connects an eligible pair v, w only if the length of $\overline{vw}$ is one of the k shortest incident on v, for some small k, or less than a small threshold value. This selection keeps the graph sparse and, if the representative points are regularly spaced, permits the edges to be unweighted without much loss of accuracy. If the distances between neighboring pairs are irregular, one may replace integer weighted edges by paths. As discussed in the previous section, one should not be so selective as to introduce a spurious disconnection in G. Verification of the base model properties is straightforward. If the graph is sparse in this scenario, the sensors will obtain much more information than the minimal amount required by property 3.

Geometric embedding, e.g. 3D gridworld: . Consider first a mobile robot with short range sensors in a 3D gridworld, e.g. an office building. This scenario is essentially the same as the 2D scenario. The 3D compass must be able to distinguish the directions "up" and "down" as well as the four 2D compass directions N,S,E,W.

We now generalize to the scenario of geometric embedding, which in a sense subsumes all three preceding scenarios. There exist a one-to-one mapping of vertices in V to points in a space, $f : V \rightarrow \Re^m$, and a mapping (usually not one-to-one) $h : E \rightarrow \Re^m$ of edges to vectors, such that $f(u) + h(\{u, v\}) = f(v)$ for all edges $\{u, v\} \in E$. An edge must be treated as an ordered pair, to avoid ambiguity in the values $h(\{u, v\}) = -h(\{v, u\})$. When the robot is at vertex v, it observes all edges incident on v, and determines, without error, the vector $h(e)$ for all such edges e. The robot might or might not determine $g(v)$, but it is certain to determine the relative locations of the vertices incident on e. Therefore the robot uniquely identifies the relative locations of its current vertex and neighboring vertices, that is, relative to the start vertex.

The base model assumptions are also consistent with many different sensor configurations and ranges, as long as the discretization is on a fine enough scale such that the sensors can observe adjacent vertices. Different vertex recognition capabilities are permitted as long as the robot can always recognize a vertex that it has observed previously. However, both actuation and sensing are assumed to be deterministic and error-free.

Upper bounds on cost for the base model apply to all consistent scenarios. This generality will simplify our derivations of performance guarantees. Lower bounds, which are proved by example, will usually have to be derived scenario by scenario, because a robot's exact behavior on a specific example depends on the details of the situation.

Some of the assumptions, such as the limited sensor range of the robot, make robots less powerful than they actually are. They allow one to show that the travel distances of greedy online robot-navigation methods are small even if the capabilities of the robot are weak and the robot-navigation tasks are therefore difficult for it. Other assumptions, such as the absence of actuator and sensor uncertainty, are somewhat simplifying but sufficiently close to reality to enable one to use the robot-navigation methods on real robots. They are approximately satisfied in structured terrain. For example, the success rate of moving a Nomad 150 mobile robot in maze-like terrain was reported to be at least 99.57 percent, and the success rate of making the correct observations in all four directions simultaneously was at least 99.38 percent. These large success rates enable one to ignore actuator and sensor uncertainty. The assumptions are also justified in less structured terrain for robots that can compensate for actuator and sensor uncertainty either with positioning systems (such as the global positioning system GPS or sensor networks) or software. For example, robots can use navigation architectures where a lower level performs local movement guided by sensory feedback and an upper level performs global navigation. The lower level can then use probabilistic methods to compensate for actuator and sensor uncertainty and the assumptions are justified for the upper level, especially in case sensor aliasing is not much of a problem because different locations tend to look sufficiently different. Given the absence or actuator and sensor uncertainty, the assumption that the robot can recognize a previously observed vertex is realistic since its can use dead-reckoning or GPS to keep track of how it moved in the terrain. The assumption is also realistic if the locations look sufficiently different.

Most of the results presented here are robust with respect to the scenario details. To give an idea of why this might be so, we discuss one particular modification of the base model here. Suppose robot A has the capabilities required by the base model, except that if it is at a vertex v it must traverse part or all of edge (v, w) to observe the neighboring vertex w. Create imaginary robot B to be identical to A, except that B possesses an extra sensor that observes all neighboring vertices from the currently occupied one. Create imaginary robot C to be identical to A, except that when C first occupies any vertex v, it executes an extra sensory subroutine that traverses each untraversed edge (v, w), thereby observing w, and returns to v. By properties 3c and 3d, robot C correctly determines that it has not occupied v previously. Similarly, it can correctly determine that it has not previously traversed an edge (v, w). Therefore the extra sensory steps involve at most $2|E|$ edge traversals.

By definition, robots B, and C each satisfy the base model. On any graph G, robot B can emulate the movements of robot A, by ignoring its extra sensor. Disregarding for now the cost of the extra sensory steps, Robot C can emulate the movements of robot B, by ignoring sensory information it acquires that B would not. Robot A can emulate the movements of C, although to do so it must execute the extra sensory steps that C would make. Since each robot can emulate the other two, at zero or modest change in cost, many results carry over with little or no change from one model to the other.

Figure 5: Mobile Robot "Xavier"

3.2 Modeling Incomplete Information

Incomplete information is typical for robots, perhaps more for mobile robots as shown in Figure 5 than for robot arms.

We now consider two common ways of handling incompleteness of (e.g. terrain) information in a discretized terrain.

- The first way is characterized by the robot knowing the general topology of the terrain, e.g. a regular grid pattern. One can think of the (initially) unknown freespace graph $G = (V, E)$ as a subgraph of a larger known graph $\hat{G} = (\hat{V}, \hat{E})$. In the case of a 2D square grid, for example, the vertices are $\hat{V} = \{0, \ldots n\}^2$, and the edges $\hat{E} = \{\{v, w\} : v \in V, w \in V, ||v - w|| = 1\}$ (with the Euclidean norm). Vertices in $\hat{V} \setminus V$ are blocked, and any edge in $\hat{G}$ that is incident on a blocked vertex is not in E, that is, $E = \hat{E} \cap \{V \times V\}$. An example is shown in Figure 8 where blocked cells are black and unblocked cells are white. The circle shows the current cell of the robot. When the robot is located at vertex v, it detects for all $e = \{v, w\} \in \hat{E}$ whether $e \in E$ and equivalently whether $w \in V$. We denote this observation using a "+" for an unblocked neighboring cell and a "-" for a blocked neighboring cell in the four compass directions north, west, south and east. In Figure 8, for example, the robot initially observes blocked cells in its west and south and unblocked cells in its east and north, denoted as "+ - - +".

  If the graph $\hat{G}$ is geometrically embedded, more topological information may be available. The robot may also know the values $f(v) : v \in \hat{V}$ or $h(e) : e \in \hat{E}$. The robot might then deduce more terrain information than the minimum required by the base model property 3. For example, if it traversed the perimeter cells of a 4 by n grid graph, it would deduce the entire graph. Without any *a priori* knowledge of the topology, i.e. of $\hat{G}$, the robot could not even deduce the graph of a 3 by n grid.

- The second way is characterized by the robot not knowing anything about the graph *a priori*, not even its topology. This is, for example, typically the case if a Voronoi graph is used to discretize the terrain. In this case, the robot generally needs a more powerful vertex identification capability. When the robot first observes a vertex, it in effect assigns a unique identifier v to that vertex. If the robot observes that vertex later, the robot recognizes it as the vertex identified as v. Thus, the only vertices and edges that exist are those of $G = (V, E)$. There is no need to augment G.

a) Grid

b) Vertex-Blocked Graph

c) Knowledge of the Robot: Known Topology

d) Knowledge of the Robot: Unknown Topology

Figure 6: *a priori* Knowledge of the Terrain

Figure 7: Alternative Vertex-Blocked Graph

The example in Figure 6 shows that it can make a difference whether a robot knows the topology of the graph *a priori*. Cells whose blockage status is unknown to the robot are grey. The circle shows the current cell of the robot. The robot moves along the arrow. If it does not know the topology of the graph in advance, it cannot, without entering the center vertex, rule out the possibility that it operates on the graph shown in Figure 7. Thus, it needs to enter the center vertex to be able to identify the graph completely. On the other hand, if it knows the topology of the graph *a priori*, it is able to identify the graph without entering the center vertex.

4 Fundamental Robot-Navigation Tasks

We study three fundamental robot-navigation tasks under incomplete information. These tasks underly many other robot-navigation tasks. Some more complex robot-navigation tasks provide the robots with even less information. SLAM (simul-

a) Example Grid

b) Vertex-Blocked Graph

c) Vertex-Blocked Graph with Blocked Vertices Removed

Figure 8: Different Kinds of Graphs

taneous localization and mapping), for example, combines localization and mapping, which is important if the robots do not have any information about the terrain or their location and cannot uniquely identify the vertices.

4.1 Mapping

Mapping means to acquire information (= map) of unknown terrain. Often, mapping means that the robot determines the graph G in which it moves. In these cases, it may start with no information or with partial information about G. For example, it may know the vertex-blocked graph $\hat{G}$ of which G is a subgraph, but not which vertices are blocked, or it may know the range of $h : E \rightarrow \Re^n$ in a geometric embedding of G.

More generally, the task of mapping is to determine as much as possible of some categories of missing terrain information. If the graph data structure $G = (V, E)$ does not contain all information about the terrain, it is possible that the robot knows the entire graph $G = (V, E)$ initially, but not some other categories of information such as the function values $f(v) : v \in V$ and $g(e) : e \in E$ of the terrain's geometric embedding. In this case the mapping task would be to determine these function values.

A robot might decide to map its terrain if its terrain changed substantially or after it was moved into new terrain. It can then exploit the map for subsequent navigation tasks. Sometimes terrain is known in principle but very difficult to model. It can then be easier to let the robot acquire it autonomously than to provide the robot with a model of it. It can, for example, be done for configuration spaces whose freespace is difficult to model, which can be the case even if the obstacles in the corresponding work spaces are simple and their placement is known. (Uninformed LRTA*, an agent-centered search method, could be used for such mapping since it eventually visits every vertex but its travel distances are too large for it to be interesting in robotics.)

4.2 Localization

Localization means to determine the current location of a robot. We assume that the robot has a map of the terrain available and even knows its orientation relative to the map (for example because it is equipped with a compass) but does not know its current location. In the model we use in this text, the robot knows the graph that it moves in, but not its current vertex. Its task is to determine its current vertex or notice that this is impossible because it is located in one of a pair of isomorphic connected graph components. We stated earlier that the robot always uniquely identifies its current vertex and explained that this capability may be achieved by different means. In localization, the robot knows the graph that it operates on but this graph is not annotated with unique vertex identifiers. Thus, the robot is not trivially localized after it observes the unique identifier of a vertex. This scenario is, for example, realistic if the robot uniquely identifies the vertices by remembering exactly how it has moved in the terrain.

A robot needs to localize after it wakes up and realizes that a user switched it off and moved it to a new location ("kidnapped robot problem"), for example to recharge it. A robot also needs to localize from time to time to verify its actual location and, if necessary, apply corrections since its control systems gradually accumulate errors due to actuator and sensor uncertainty. In this context, localization eliminates the need for complex and expensive positioning systems, for example based on radio beacons, inside of buildings, in streets with tall buildings or on other planets, where three satellites are not in view and thus GPS is not effective.

4.3 Goal-Directed Navigation in Unknown Terrain

Goal-directed navigation requires that there be a geometric embedding of the terrain. The problem is to move the robot to given goal coordinates in *a priori* unknown terrain. The coordinates may be absolute, or they may be relative to the start vertex. In the model we use in this text, the robot knows its current vertex and the topology of a vertex-blocked graph but not which vertices are blocked. Its task is to reach a given vertex (= the goal vertex). (Informed or uninformed LRTA* could be used for goal-directed navigation in unknown terrain but its travel distances are again too large for it to be interesting in robotics.)

A robot must navigate to a goal location in *a priori* unknown terrain, for example, to check out a given location for survivors after an earthquake. In this case, it is sufficient to map as much of the terrain as necessary to move the robot to the goal location. It is unnecessary and often too time-consuming to map the terrain completely.

5 Planning Objective

It is important to empirical robotics researchers that their robot-navigation methods plan in real-time and result in small travel distances and thus also small plan-execution times and, since planning is fast, small task-completion times. We describe a worst-case analysis of the travel distance because a robot-navigation methods with a small worst-case travel distance always perform well, which matches the concerns of empirical robotics researchers for guaranteed performance.

One could simply test robot-navigation methods empirically to determine whether their travel distances are small. However, they need to get analyzed theoretically to guarantee that their travel distances are small in any kind of terrain to rule out that they are small in empirical tests only because of properties of the test terrain.

The travel distance of a robot-navigation method depends on not only the robot-navigation method itself but also the terrain, the start location of the robot in the terrain, the tie-breaking strategy used to decide between seemingly equally good navigation choices, and so on. We determine the travel distance in the worst case and state for each robot-navigation method over which quantities we calculate the worst case. We then describe upper and lower bounds on their worst-case travel distance on graphs of a given size, measured by their number of vertices. This allows one to determine whether the worst-case travel distance of a robot-navigation method is minimal and, if not, how suboptimal it is, by comparing these bounds against the best possible worst-case travel distance of any robot-navigation method that has the same *a priori* information.

Researchers sometimes use on-line rather than worst-case criteria to analyze robot-navigation methods. For example, they are interested in robot-navigation methods with a small competitive ratio. The competitive ratio compares the travel distance of a robot to the distance that an omniscient robot that has complete *a priori* information would need to move only to verify that information. (For example, the robot already knows its current location for localization problems and only needs to verify it.) Minimizing the ratio of these quantities minimizes regret in the sense that it minimizes the value of k such that the robot could have localized k times faster if it had already known its location. The competitive ratio has little relation to worst-case performance if robots do not have complete *a priori* information. Furthermore, the difference in travel distance between robot-navigation methods that do and do not have complete information is often large. For goal-directed navigation, for example, the robot can follow a shortest path from the start cell to the goal cell if it knows the terrain but usually has to try out many promising paths on average to get from the start cell to the goal cell if it does not know the terrain.

6 Planning

The sensors on-board a robot can typically sense terrain only near its current location. The robot thus has to move in the terrain to sense new parts of it, either to discover more about the terrain or its current location. Therefore, a robot has to find a reconnaissance plan that determines how it should move to make additional observations, which is called sensor-based planning. The plan is a conditional plan that can take into account all of the information that the robot has already gathered (namely, the sequence of moves it has executed and the sequence of observations it has made in return) when determining the next move of the robot. A deterministic plan specifies the move directly while a randomized plan specifies a probability distribution over the moves. No randomized plan that solves one of the robot-navigation tasks has a better worst-case expected travel distance (where the expectation is taken with respect to the randomized choice of moves by the randomized plan) than a deterministic plan with minimal worst-case travel distance. In this text, we therefore consider only deterministic plans.

a) Example Grid

b) Knowledge of the Robot

Figure 9: Greedy Localization

The robot can either first determine and then execute a complete plan (off-line planning) or interleave partial planning and moves (on-line planning). We now discuss these two options in greater detail, using localization tasks as example. Localization consists of two phases, namely hypothesis generation and hypothesis elimination. Hypothesis generation determines all locations that are consistent with the observation made by the robot in its start location. It simply involves going through all vertices and eliminating those that are inconsistent with the observation. (There also exists a more complex but polynomial time hypothesis generation method for continuous polygonal terrain.) If the set contains more than one location, hypothesis elimination then tries to determine which location in this set is the true location of the robot by moving the robot and eliminating all locations that are inconsistent with the observations made by the robot. The hypothesis-elimination method involves planning, which we discuss in the following.

6.1 Optimal Off-Line Planning

Off-line planning finds a complete and thus potentially large plan before the robot starts to move. A complete localization plan is valid if and only if it eventually correctly determines either the current vertex of the robot or the fact that the current vertex of the robot cannot be determined uniquely, no matter which unblocked vertex is the start vertex of the robot.

Consider, for example, the localization task from Figure 9 and the corresponding state space from Figure 10 that is reachable from the start state. The states are sets of cells, namely the cells that the robot could be in. The robot initially observes "+ - - -". The question marks indicate the cells that are consistent with this observation, namely E2, E4 and E6. The start state thus contains these three cells. (The robot can rule out B7 since it has a compass on-board.) Every state that contains only one cell is a goal state. In every non-goal state, the robot can choose a move ("OR" nodes of the state

Figure 10: Part of the State Space for Localization

Figure 11: Worst-Case Optimal Localization Plan

space), described as a compass direction. It then makes a new observation ("AND" nodes of the state space). The state space is non-deterministic since the robot cannot always predict which observation it makes and thus which effects its future moves have. For example, after moving north two times from the start state, it reaches a state that contains three possible cells, namely C2, C4 and C6. When moving north in this state, the robot could observe "- - + +" and then be in the state that contains only the cell B2. It then cannot move to the west even if this state wasn't a goal state. It could also observe "- + - +" and then be in the state that contains the cells B4 and B6. It then can move to the west.

Deterministic localization plans assign a move to each OR node in the state space. Valid deterministic localization plans with minimal worst-case travel distance can be found with complete AND-OR searches, resulting in decision trees since states cannot repeat in worst-case optimal plans. Such a decision tree is shown in Figure 11. However, it is intractable to perform complete AND-OR searches. This is not surprising since the states are sets of cells and their number is thus large (although not all sets of cells can appear in practice). In fact, we discuss later that finding valid localization plans with minimal worst-case travel distance is NP-hard and thus very likely needs exponential planning time, which is consistent with results by the theoretical planning community that show the complexity of planning with incomplete information to be high in general.

6.2 Greedy On-Line Planning

One needs to speed up planning by developing robot-navigation methods that sacrifice the optimality of the resulting robot paths to achieve real-time planning. If done correctly, the suboptimality of the robot path and thus the increase in plan-execution time are outweighed by the decrease of the planning time so that the sum of planning and plan-execution time (that is, the task-completion time) decreases substantially. For robot-navigation tasks with incomplete information, this can be done by interleaving planning and moves to gather information about the terrain early and then use the acquired information for re-planning right away. The acquired information makes subsequent planning faster since it reduces the uncertainty of the robot about the terrain or its location, which reduces the amount of planning performed for unencountered situations and, more generally, the size of the state space reachable from its current state.

Planning in deterministic state spaces is fast. Even with a data structure that is no more sophisticated than a binary heap, a shortest path in a graph can be found in time $O(|E| \log |V|)$ for arbitrary graphs $G = (V, E)$ and $O(|V| \log |V|)$ for planar graphs $G = (V, E)$, which includes grid graphs. Greedy on-line robot-navigation methods make use of this property to solve planning tasks in non-deterministic state spaces by interleaving myopic planning in deterministic state spaces and moves (= on-line planning). They determine plans under the (wrong) assumption that robots do not gain additional information during plan execution and thus do not take the long-term consequences of the robot moves into account in case they do gain additional information, resulting in greedy planning. The result is a trade-off in planning and plan-execution time.

We now introduce two greedy on-line planning techniques, namely agent-centered search and assumption-based planning. They differ in how they make planning deterministic.

- Agent-centered search methods (in non-deterministic state spaces) plan with limited lookahead by performing partial AND-OR searches, forward from the current state, instead of a complete one. They restrict planning to the part of the state space around the current state (local search), which is the part of the state space that is immediately relevant for the robot in its current situation because it contains the states that the robot will soon be in. Thus, agent-centered search methods decide on the part of the state space to search and then determine how to move within it. Then, they execute these moves (or only the first move) and repeat the overall process from the resulting state, until the planning task is solved. Consequently, agent-centered search methods thus avoid a combinatorial explosion by finding only prefixes of complete plans.

  Agent-centered search methods can be implemented with real-time heuristic search and use techniques tai-

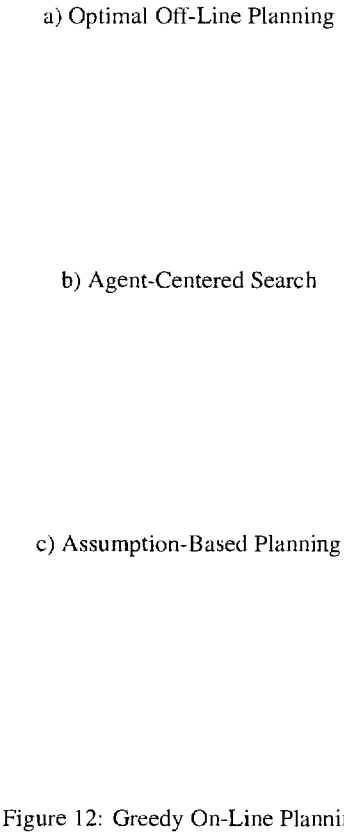

Figure 12: Greedy On-Line Planning

lored to agent-centered search methods or more general techniques from limited rationality and deliberation scheduling to determine how much to plan. One needs to avoid infinite cycles when not planning all the way from the current state of the robot to a goal state. The agent-centered search methods that we describe in this text do this by making the lookahead large enough so that the subsequent plan execution results in a gain of knowledge, although real-time heuristic search methods could be used to get away with smaller lookaheads. The agent-centered search methods that we describe in this text use a simple approach to make planning fast, namely by planning exactly in the deterministic part of the non-deterministic state space around the current state, with the objective to execute a non-deterministic move and thus gain information quickly. This is the part of the state space in Figure 10 that has a border around it. The robot then executes the plan, the last move of which is non-deterministic. It then observes the resulting state and repeats the process from the state that actually resulted from the move instead of all states that could have resulted from it, until the planning task is solved.

Figure 12 b illustrates agent-centered search. The large triangle represents the state space of an optimal off-line search method while the shaded areas represent the state spaces of agent-centered search, one for each planning episode, that is, planning performed between moves. The shaded areas together are much smaller than the area of the large triangle but the resulting path is not minimal. Examples of agent-centered search methods are Greedy Localization and Greedy Mapping, two robot robot-navigation methods for localization and mapping, respectively.

- Assumption-based planning methods, on the other hand, plan all the way from the current state to a goal state but make assumptions about the move outcomes. Therefore, assumption-based planning methods find a complete plan from the current state to the goal state and execute the plan. If a move results in an outcome that was ignored during planning, then they repeat the process from the state that resulted from the move, until the planning task is solved. Consequently, assumption-based planning methods avoid a combinatorial explosion because they ignore some outcomes of moves and thus do not plan for all contingencies, in a way similar but not identical to envelope-based planning.

  A simple approach that makes planning fast is to assume that the state space is deterministic by ignoring all outcomes of each move but one.

  Figure 12 c illustrates assumption-based planning. The shaded areas represent the state spaces of assumption-based planning, one for each planning episode. The shaded areas together are much smaller than the area of the large triangle but the resulting robot path is not minimal. An example of an assumption-based planning method is Planning with the Freespace Assumption, a robot-navigation method for moving to a goal location in unknown terrain.

Greedy on-line robot-navigation methods are common-sense robot-navigation methods that have often been discovered and implemented by empirical robotics researchers. In the remainder of the text, we study the planning time and resulting travel distance of Greedy Mapping, Planning with the Freespace Assumption and Greedy Localization. We demonstrate that these three greedy on-line robot-navigation methods plan in low order polynomial time and have small worst-case travel distances.

7 Greedy Mapping

Greedy Mapping uses agent-centered search to map unknown terrain. It always moves the robot on a shortest path from its current vertex to a closest informative vertex until it cannot reach any informative vertex any longer. An informative vertex is a vertex at which the robot can acquire new information for its map.

Our definition of Greedy Mapping may seem unnecessarily vague, but the generality is quite deliberate. There are many kinds of maps that the robot could seek to acquire, and many possible states of prior knowledge. The most obvious map to seek is the graph $G = (V, E)$ of the base model. In this case, the robot might initially know nothing of G, or know $\hat{G}$ but not which vertices are blocked, or know the pattern of a geometric embedding of G, etc. In other cases, the robot might

a) Example Grid

b) Knowledge of the Robot

Figure 13: Greedy Mapping

instead know G *a priori*, and seek a map of the geometric embedding of G. It might seek a visibility map that for all v and $w \in V$ records whether or not w can be observed from v, or for all $v \in V$ and $e \in E$ records whether or not edge e can be observed from v.

Regardless of what other information it collects, Greedy Mapping is sure to maintain the partial graph that the robot has learned so far, relative to its start vertex. This minimal amount of information gathering is guaranteed by the base model properties. We clarify a somewhat subtle issue in case the robot does not know the topology of the graph *a priori*. Every closest informative vertex in the graph is also in the currently known subgraph. (Let w denote an informative vertex closest to the currently occupied vertex v'. Let v be the vertex adjacent to w on a shortest path in the graph from v' to w. By definition of w, vertex v cannot be informative because v is closer to v' than is w. Therefore, the robot possesses all the information it would obtain at v. This information by property 3 includes the observation of w and the edge (v, w), whence w is currently known.) Therefore, there is no harm in defining Greedy Mapping in terms of closest vertices in the graph rather than the currently known subgraph even if the topology of the graph is not known.

Figure 13 illustrates the behavior of Greedy Mapping if the topology of the graph is known *a priori*. Arrows show the paths planned by Greedy Mapping whenever Greedy Mapping determines them. The robot then moves along such a path without planning, and an arrow is thus only shown for the first move of each path.

Greedy Mapping moves after each planning episode to a closest informative vertex, which by property 4 becomes and remains uninformative. Hence here are at most $|V|$ planning episodes. Each episode finds a shortest path of length at most $|V|$, in at most $O(|V|^2)$ planning time. Therefore Greedy Mapping requires at most $|V|^2$ moves and at most $O(|V|^3)$ planning time, in total. After termination, Greedy Mapping has learned everything about the terrain that it possibly can since all vertices in the graph have become uninformative.

Greedy Mapping is a common-sense robot-navigation method that has been discovered and implemented independently by several empirical robotics researchers. It has been used on a nomad-class tour-guide robot that offered tours to museum visitors. It has also been used on Nomad 150 mobile robots and Super Scouts.

Greedy Mapping use deterministic search methods because it performs an agent-centered search in exactly all of the deterministic part of the non-deterministic state space, namely exactly up to the point where it executes a non-deterministic move and thus gains more information about the graph. Figure 14 shows the state space that is reachable from the current state of the mapping problem when the robot reaches cell B2 in Figure 13. It is exponential in the number of vertices since the states are pairs of vertices and partial graphs, namely the current robot vertex and the currently known subgraph. If the robot moves within the known portion of the terrain, the robot can predict the resulting successor states with certainty and the moves thus have deterministic effects. Otherwise, the moves have several possible outcomes and thus non-deterministic effects. Observing the actual outcome results in a gain in information. Greedy Mapping determines the shortest move sequence that reaches the fringe of the deterministic part of the state space, which has a border around it in the figure, and thus ends in a non-deterministic move.

Greedy Mapping has to re-plan frequently. It can calculate the shortest paths efficiently with incremental heuristic search methods, such as Dynamic A* or the simpler D* Lite. They combine two principles for searching efficiently, namely heuristic search (using approximations of the goal distances to focus the search) and incremental search (re-using information from previous searches to avoid recomputations of quantities that have not changed since the previous search). While heuristic search is well known and often described in textbooks on both artificial intelligence and robotics in the context of A* , incremental search is not as well known as it was pioneered in algorithm theory. Incremental search has the potential to speed up planning by re-using information from previous searches. Figure 15 illustrates the idea behind incremental search in the context of Greedy Mapping, where the numbers in cells that are known to be unblocked are their goal distances, that is, how many moves is the nearest informative cell away from the given cell. Incremental search first determines the goal distances.

Figure 14: Part of the State Space for Mapping

The robot then automatically moves along a shortest path from its current cell to a closest informative cell when repeatedly moving from its current cell to a neighboring cell with a smallest goal distance. The advantage of determining the goal distances first rather than the shortest path directly is that typically only very few goal distances change between searches while the path can change completely, as shown in Figure 15 when the robot is in cell C2 for the first time and then moves to cell B2. The path changes completely but only two goal distances change. Incremental search calculates only those goal distances that have changed since the immediately preceding search (incremental search) *and* that

Figure 15: Re-Planning for Greedy Mapping

are important for the current search (heuristic search). They can be understood as transforming the A* search tree of the previous search problem into the A* search tree of the new search problem which is more efficient than constructing the new search tree from scratch if both search trees are similar.

7.1 Qualitative Advantages

While it is important to empirical robotics researchers that robot-navigation methods plan in real-time and the resulting travel distances are sufficiently small, other properties of the robot-navigation methods are also important to them. Some of these properties are hard to characterize and quantify and, in general, are only poorly understood. Thus, they are seldom taken into account when theoretical robotics researchers design robot-navigation methods although they can provide important design considerations. In the following, we list some advantages of Greedy Mapping that are shared by other greedy on-line robot-navigation methods. These advantages result from the fact that greedy on-line robot-navigation methods can re-plan after every move of the robot, which makes them reactive to changes in state such as robot location and terrain information.

- **Reactive to Location Changes:** Greedy Mapping is reactive to changes in the location of the robot since it can re-plan after every move, making use of all of the information the robot has available about its location. Thus, it does not predict the current location of the robot based on its move recommendations but rather uses the actual location of the robot. This is advantageous because the robot might not have followed the move recommendation of Greedy Mapping, due to interventions by other modules of the robot architecture, or might not have been able to execute the move recommendation success-

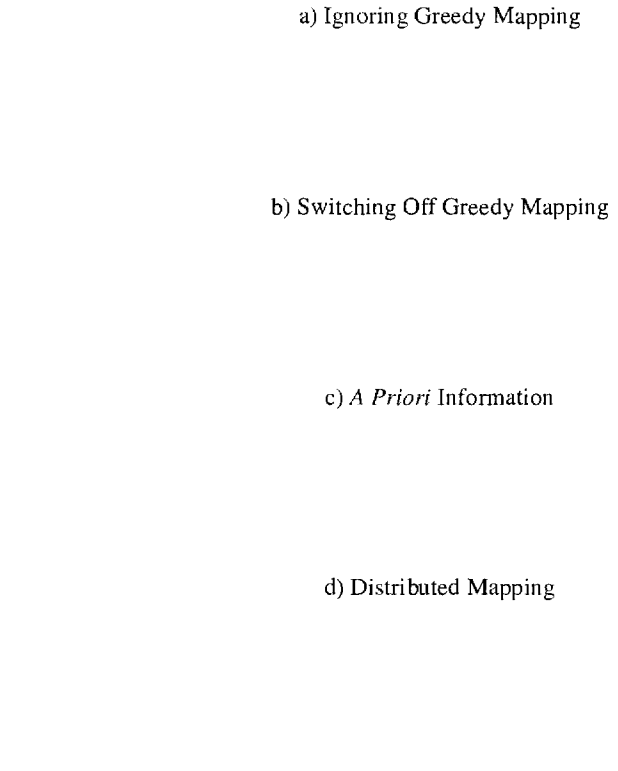

Figure 16: Advantages of Greedy Mapping

fully, due to external causes. Being reactive to the actual location of the robot makes it easy to integrate Greedy Mapping into complete robot architectures because it does not need to have control of the robot at all times, implying that it can be implemented as one module of a robot architecture, such as a subsumption architecture, and is also robust with respect to the inevitable inaccuracies and malfunctions of other architecture components.

- **Ignoring Greedy Mapping:** One consequence of this property is that Greedy Mapping can easily co-exist with other modules of a robot architecture that might alter its move recommendations from time to time. This is important because search methods should only provide advice on how to act and work robustly even if that advice is ignored from time to time. For example, if Greedy Mapping suggests to pass an obstacle very closely to minimize the travel distance, its move recommendation might get changed by the obstacle-avoidance module to pass the obstacle with a larger safely margin. This is not a problem for Greedy Mapping since it automatically resumes its operation from the new location of the robot, as shown in Figure 16 a.
- **Switching Off Greedy Mapping:** Another consequence of this property is that Greedy Mapping can easily co-exist with other modules of a robot architecture that might ignore its move recommendations from time to time. For example, if a robot has to recharge its batteries during mapping, then an exception-handling module might have to preempt mapping and move the robot to a known power outlet. Once restarted, it would be cumbersome to return the robot to the location where mapping was stopped (which could be far away) and resume its operation from there. This is not a problem for Greedy Mapping since it automatically resumes its operation from the power outlet, as shown in Figure 16 b.

- **Reactive to Knowledge Changes of the Terrain:** Greedy Mapping is also reactive to changes in the knowledge that the robot has of the terrain since it can re-plan after every move, making use of all of the information the robot has available about the terrain. It takes new information into account immediately when it becomes available and can change the path of the robot right away to reflect its new knowledge. It does not matter how this information was obtained, for example whether this information was learned by the robot or obtained from other sources.
  - *A Priori* **Information:** One consequence of this property is that Greedy Mapping can take advantage of *a priori* terrain information, if available, in case part of the terrain is already known and thus does not need to get mapped, for example via satellite reconnaissance or prior partial exploration, as shown in Figure 16 c.
  - **Distributed Mapping:** Another consequence of this property is that Greedy Mapping can take advantage of terrain information obtained on-line by other robots, if available. Several researchers have developed ways how several robots can map terrain cooperatively, thereby decreasing the mapping time. Mapping tasks can also be solved with several robots cooperatively that each run Greedy Mapping and share their maps, as shown in Figure 16 d. In practice, one has to use somewhat more sophisticated versions of Greedy Mapping to make nearby robots move in different directions.

7.2 Alternative Approach

Depth-first search (that is, chronological backtracking) can also be used to map unknown terrain. Depth-first search always moves the robot from its current vertex to a neighboring vertex that is unvisited. If such a vertex does not exist, it leaves the current vertex along the edge with which it was entered for the first time (backtracking). It terminates when it knows of no unvisited vertices. Figure 13 serves also as a partial example of the behavior of depth-first search if the topology of the graph is known *a priori*. However, depth-first search has none of the qualitative advantages of Greedy Mapping. For example, depth-first search does not resume mapping from the power outlet after a robot has moved to recharge itself but rather requires the robot to return to the location where mapping was stopped. While it may be the case that simple depth-first search can be modified to acquire some

Figure 17: Lower Bound on Greedy Mapping

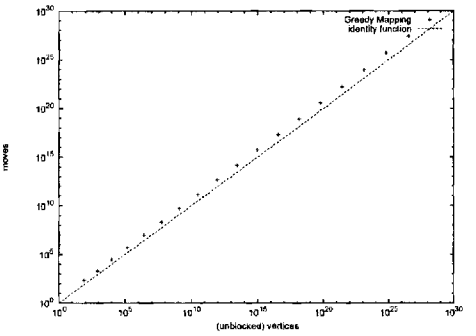

Figure 18: Increase of Travel Distance

or all of the qualitative advantages of Greedy Mapping while retaining its other advantages, these attempts have so far resulted in impractical robot-navigation methods. Greedy Mapping can also be generalized easily to continuous polygonal terrain whereas it is unknown how to extend depth-first search to polygonal terrain without discretizing the terrain first into a graph.

7.3 Empirical Travel Distance

Experimental results show that the average-case travel distances of Greedy Mapping and depth-first search tend to be approximately linear in the number of (unblocked) vertices, with the travel distance of Greedy Mapping being smaller than the travel distance of depth-first search. This result holds both for grid graphs with random obstacles and maze-like grid graphs whose corridors were created by a randomized depth-first search with some additional cells being made unblocked. (These results were actually obtained for the "Closest Unvisited Vertex" version of Greedy Mapping, see below.) However, these experimental results could have been the result of properties of the test terrains. The worst-case travel distance of Greedy Mapping is not necessarily minimal because, due to its greedy nature, it might not visit nearby vertices and then has to return later to visit them. An example is given in Figure 13, where Greedy Mapping could easily have moved to cell E3 from cell E2 and later from cell D3 to observe the blockage status of cell E4 but did not.

7.4 Lower Bound on Worst-Case Travel Distance

To understand how bad the travel distance of Greedy Mapping can be, we analyze the worst-case travel distance of Greedy Mapping as a function of the number of vertices of the graphs, where the worst case is taken over all graphs of the same number of vertices, over all possible start vertices of the robot and over all possible tie-breaking strategies used to decide which one of several equally close informative vertices to visit next. We describe a lower bound on the worst-case travel distance of Greedy Mapping in this section and an upper bound in the next section.

Lower bounds on the worst-case travel distance of Greedy Mapping are established by example. We first study Greedy Mapping when the robot does not know the topology of the graph *a priori*. If the sensors provide only the minimum base model capabilities (3), then every unvisited vertex is informative. Even if vertex v and all edges incident on v have been observed, because all of v's neighbors have been visited, the robot can not determine that there are no other hitherto unobserved edges incident on v without also visiting v. The worst-case travel distance of Greedy Mapping then is at least linear in the number of vertices even on planar graphs since every vertex must be visited. We now prove a larger lower bound by presenting an example graph that is planar since graphs used in robotics often have this property. It consists of a long horizontal rim and a set of spokes of varying lengths attached to the rim at various vertices, as shown in Figure 17. Each spoke consists of a pair of parallel paths of the same length that connect the rim to a single edge. The leaf vertices at the ends of these single edges are crucial to "fooling" Greedy Mapping into traversing the rim repeatedly forward and backward since Greedy Mapping might choose to return to the rim along the other path without first exploring the leaf vertex when the robot traverses one of the parallel paths. The resulting travel distance of Greedy Mapping is large compared to the number of vertices for the spokes that are necessary to mislead Greedy Mapping into this robot path, resulting in Theorem 1. This worst-case travel distance is slightly superlinear in the number of vertices, as shown in Table 2 and the log-log graph in Figure 18, since $\log |V|$ increases only very slowly in $|V|$ and $\frac{\log |V|}{\log \log |V|}$ increases a trifle more slowly in $|V|$.

Theorem 1 *On the class of planar graphs $G = (V, E)$, the worst-case travel distance of Greedy Mapping is $\Omega(\frac{|V| \log |V|}{\log \log |V|})$.*

Proof: Consider graphs of the kind shown in Figure 17. The rim has length n^n for some integer $n \geq 3$ and consists of the vertices $v_0, v_1, \ldots, v_{n^n}$. We define the length of a spoke to be the length of each of its two paths. For each integer i with $1 \leq i \leq n$ there are n^{n-i} class i spokes of length $\sum_{j=1}^{i-1} n^j$ each (including class 1 spokes of length zero), as shown in Table 1. These spokes attach to the rim at vertices v_{jn^i} for integers j; if i is even, then $0 \leq j \leq n^{n-i} - 1$, otherwise $1 \leq j \leq n^{n-i}$. There is one additional single edge that attaches to vertex v_0, leading to a tail vertex. The start vertex is v_{n^n}. Greedy Mapping can choose to break ties so as to behave as follows (for any tie-breaking rule the example graph could be modified to enforce this robot path): start at

Table 1: Spokes of the Example Graph

number of spokes	length of each spoke	vertices at which the spokes attach to the rim
n^{n-1}	0	$v_n, v_{2n}, v_{3n}, \ldots, v_{n^n}$
n^{n-2}	n	$v_0, v_{n^2}, v_{2n^2}, \ldots, v_{n^n-n^2}$
n^{n-3}	n^2+n	$v_{n^3}, v_{2n^3}, v_{3n^3}, \ldots, v_{n^n}$
n^{n-4}	n^3+n^2+n	$v_0, v_{n^4}, v_{2n^4}, \ldots, v_{n^n-n^4}$
$\ldots$	$\ldots$	$\ldots$

Table 2: Possible Travel Distance of Greedy Mapping on the Example Graph

n	moves	$\lvert V\rvert$	$\frac{\text{moves}}{\lvert V\rvert}$
3	207	80	2.587500
4	2279	778	2.929306
5	31253	9612	3.251457
6	515085	144014	3.576631
7	9928271	2542528	3.904882
8	219130987	51744018	4.234905
9	5448100629	1193201300	4.565953
10	150617283953	30753086422	4.897631
11	4582105430015	876166203024	5.229722
12	152089515084303	27343936154330	5.562093
13	5468578313471236	927716806196572	5.894663
14	211719893362703040	33998232210572476	6.227379

vertex v_{n^n}, traverse the whole rim and all spokes, but bypass all the leaf vertices at the spoke ends, and then traverse the additional edge attached to vertex v_0, as shown in Figure 17. At this point, Greedy Mapping again traverses the whole rim, visiting the leaf vertices of the class 1 spokes of length 0. It then switches directions and moves along the whole rim in the opposite direction, this time visiting the leaf vertices of the spokes of class 2 (length n), and so forth, traversing the rim once for each class of spokes.

The key property of the construction is that whenever the robot has returned to the rim from a class i spoke, the leaf of the next nearest class i spoke is no farther than the length of a class $i+1$ spoke. Hence Greedy Mapping may visit all class i spoke leaves before visiting any class $i+1$ spoke leaves. To summarize, the leaf vertices at the ends of the spokes are tried out in the order indicated in Figure 17. The travel distance is $\Omega(n^{n+1})$ moves since the rim of length n^n is traversed $n+1$ times. To be precise, the travel distance is $(n^{n+3}+3n^{n+2}-8n^{n+1}+2n^2-n+3)/(n^2-2n+1)$ moves. It holds that $|V| = \Theta(n^n)$ since $|V| = (3n^{n+2} - 5n^{n+1} - n^n + n^{n-1} + 2n^2 - 2n + 2)/(n^2 - 2n + 1)$. This implies that $n = \Omega(\frac{\log |V|}{\log \log |V|})$ since

$$\frac{\log n^n}{\log \log n^n} = \frac{n \log n}{\log n + \log \log n} \leq \frac{n \log n}{\log n} = n.$$

It follows that the travel distance of Greedy Mapping is $\Omega(n^{n+1}) = \Omega(n\,|V|) = \Omega(\frac{V|\log|V|}{\log\log|V|})$. ■

How meaningful is this super-linear bound on Greedy Mapping's worst-case performance? For one thing, the example graph is somewhat unrealistic and the worst-case travel distance of Greedy Mapping might be smaller than the lower bound on more realistic graphs. For another, the discretization methods presented earlier often assure that the graph topology is highly structured and known in advance. Might this information permit Greedy Mapping to perform better in the worst case?

Figure 19: Each Spoke Class Bends at a Distinct Height

Unfortunately, it turns out that the same order super-linear lower bound also applies to Greedy Mapping when the robot knows the topology of the graph. We prove this by transforming the example graph into a 2D grid graph. As a first step, place spokes of class i at vertices $v_{jn+3(i-1)}$. Extend the length of the rim slightly as needed. This offsets all class i spokes by the same amount, $3(i-1)$, so that no rim vertex

Figure 20: Spoke Detail, n=4

Figure 21: Bent Spoke, $n = 12$, Detail

need have impossibly many neighbors for a grid graph. Since offsets are the same within a class, the key property of the construction used in the preceding proof is retained (in fact strengthened by 2). For the second step, represent each vertex by a grid graph cell. The rim will be a horizontal row of cells. A spoke of length L, attached at v_t, consists of a vertical column of length $L + 1$ attached to the rim at v_t, and a parallel one of length L attached at v_{t+1}. Figure 7.4 illustrates the class 0 and 1 spoke constructions for $n = 4$. The $L + 1$st vertex of the first column is the spoke's leaf. (We need not alternate the leaf's location by the parity of i, because the key property is stronger by 2 in this construction.) The third and final step makes for more efficient use of space, though it is not necessary, strictly speaking. Bend the spokes so that most of their length utilizes horizontal space. Figure 7.4 illustrates this step conceptually; Figure 7.4 illustrates the step in detail for class 2 spokes with $n = 12$. This step greatly reduces the rectangular area needed to contain the graph, from $O(n^{2n-1})$ to $O(n^{n+1})$. This grid graph again fools Greedy Mapping into traversing all but a negligible portion of the lengthy rim n times. However, the resulting grid graph is too large to make a good testbed for testing mapping methods empirically.

The lower bound can be adapted easily to other versions of Greedy Mapping and other robot capabilities. For example, the freespace of any 2D grid graph is, literally, a polygon with polygonal obstacles, freespace moreover that has been conveniently discretized into cells. The number of line segments is, also conveniently, $O(|V|)$. Suppose the robot has long-range sensors, as in the continuous polygonal scenario defined previously, that are operated from the center of the cell the robot is in. (We could instead permit the sensors to be operated from every point in the interior of the cell). The robot equipped with these more powerful sensors would greedily map the grid graph in time $O(|V|)$. Modify the graph by replacing all straight corridors with twisty corridors that prevent the long range sensors from seeing far. This modification scales the graph by only a constant factor, and achieves the same order lower bound as before.

7.5 Upper Bound on Worst-Case Travel Distance

The example graph from Figure 17 provides a lower bound on the worst-case travel distance of Greedy Mapping. There might exist other graphs that result in even larger worst-case travel distances. The upper bound presented next proves that the lower bound cannot be far from tight. We have shown earlier that the worst-case travel distance of Greedy Mapping is at most $|V|^2$ moves. This obvious upper bound was decreased first to $O(|V|^{3/2})$ moves and then to the one given in Theorem 2. We first describe a general result about greedy agent-centered search methods which can be used to prove the upper bound. The proof of the general result uses the same ideas of bounding the number of planning episodes and the travel distance between planning episodes that justify the quadratic upper bound on the worst-case travel distance of Greedy Mapping. These upper bounds have been used by empirical robotics researchers to justify their choice of Greedy Mapping.

We define the behavior of a greedy agent-centered search method on a graph under the assumptions from Section 3.1 as follows: Initially, all vertices are unmarked. The greedy agent-centered search method always marks its current vertex, and perhaps other vertices as well. A marked vertex always remains marked. The search method repeatedly moves the robot from its current vertex on a shortest path to a closest unmarked vertex until all vertices are marked. We make no

assumptions regarding which vertices are marked as in addition to the current vertex nor regarding how ties are broken between several equally close unmarked vertices.

Lemma 1 *The worst-case travel distance of any greedy agent-centered search method is at most* $|V| + 2|V| \ln |V|$ *moves on graphs* $G = (V, E)$.

The logarithm in the lemma is a natural logarithm.

Theorem 2 *If the base model properties hold, then the worst-case travel distance of Greedy Mapping is* $O(|V| \log |V|)$ *moves on arbitrary graphs* $G = (V, E)$*, regardless of what information is sought for the map, and what information is known a priori.*

Proof: If one defines an informative vertex to be unmarked, then Greedy Mapping is a greedy agent-centered search method. In particular, property 4 assures that marked vertices remain marked. Lemma 1 then states that its worst-case travel distance is at most $|V| + 2|V| \ln |V|$ moves on graphs $G = (V, E)$. ■

Variants of greedy mapping

The upper bound of Theorem 2 holds for the base model, and therefore applies to a wide range of situations, including different amounts of *a priori* information and different sensor types or ranges. The key properties are deterministic sensing, so that uninformative vertices remain uninformative (4), and that the sensors are powerful enough to observe all neighboring vertices (3).

The upper bound also holds for several variants of Greedy mapping. Recall that a vertex $v \in V$ is informative if Greedy Mapping can gain new information about the map when being at v. We define a vertex $v \in V$ to be scanned if and only if Greedy Mapping has observed it, all its incident edges, and all its neighboring vertices. In terms of the base model, v is scanned if the robot has learned the minimum information that must be acquired at v. Different variants of Greedy Mapping all employ the same basic principle: always move the robot from its current vertex on a shortest path toward a closest unmarked vertex. A vertex $v \in V$ is unmarked if it is still unvisited, unscanned or informative - depending on the version of Greedy Mapping. Once the robot has been at v, that vertex becomes visited, scanned and uninformative and remains that way, because of property 4.

Subtleties arise if unanticipated information may be received en route. An unmarked vertex might become marked before it is reached. Another complication may occur if sensor range varies from vertex to vertex but the extent of the range but not informative about terrain. It may not be possible to determine whether a vertex is informative without visiting it. The upper bound often extends to these scenarios but requires a more complicated proof.

Closest Unvisited Vertex: This version of Greedy Mapping always moves on a shortest path from its current vertex to a closest unvisited vertex and repeats the process when it reaches that vertex, until it cannot reach any unvisited vertex any longer.

Closest Unscanned Vertex: If the robot has longer-range sensors, it may be able to scan a vertex from a distance, without visiting it. Therefore, Greedy Mapping might not have to visit every vertex. This version of Greedy Mapping therefore always moves on a shortest path from its current vertex to a closest unscanned vertex and repeats the process when it reaches that vertex, until it cannot reach any unscanned vertex any longer.

Closest Unscanned Vertex with Re-Planning: This version of Greedy Mapping is the same as the version of Greedy Mapping that moves toward the closest unscanned vertex, except that it immediately repeats the process instead of continuing on to its target vertex if it is able to scan the vertex before it reaches it. The resulting behavior is equivalent to repeatedly traversing the first edge of a shortest path from the current vertex to a closest unscanned vertex.

Closest Informative Vertex with Re-Planning: This version of Greedy Mapping always moves on a shortest path from its current vertex towards a closest informative vertex. It may receive additional information exogenously at any time, e.g. from other robots or devices. The informational status of a vertex may change while the robot is en route, forcing re-planning.

Closest Informative Vertex: This version of Greedy Mapping always moves on a shortest path from its current vertex to a closest informative vertex and repeats the process when it reaches that vertex, until it cannot reach any informative vertex any longer. This version of Greedy Mapping corresponds is the one that we have have used throughout this report.

If the robot does not know the topology of the graph *a priori*, then the concepts of unvisited unblocked vertex, unscanned unblocked vertex and informative vertex are identical. Thus, all versions of Greedy Mapping result in the same behavior of the robot (modulo tie breaking). Re-planning does not make a difference since a closest unvisited, unscanned or informative unblocked vertex is the first such vertex encountered. If the robot knows the topology of the graph *a priori* but not which vertices are blocked, then only the "Closest Unscanned Vertex with Re-Planning" and "Closest Informative Vertex" version of Greedy Mapping result in the same behavior of the robot (modulo tie breaking) if the sensor range of the robot is only one cell. (We leave it to the reader to verify these statements.) The proofs that the worst-case travel distance of the "Closest Unvisited Vertex" and "Closest Unscanned Vertex" version of Greedy Mapping is $O(|V| \log |V|)$ moves on graphs $G = (V, E)$ follows directly from Lemma 1, in the same way as the proof of Theorem 2 for the "Closest Informative Vertex" version of Greedy Mapping, if we define unvisited unblocked and unscanned unblocked vertices to be unmarked. The proof for the "Closest Vertex with Re-Planning" versions is different because Greedy Mapping might not follow the planned path completely.

7.6 Discussion of the Results

No mapping method can guarantee to omnisciently follow a best possible robot path in hindsight. To judge how good the worst-case travel distance of Greedy Mapping is, one therefore needs to compare it to other mapping methods that can

be used in its place, such as depth-first search. The worst-case travel distance of depth-first search, even if it does not know the topology of the graph *a priori*, is at most twice the number of vertices since each move either visits a still unvisited vertex (which can happen at most once for each vertex) or backtracks from a vertex (which can also happen at most once for each vertex). No robot-navigation method can do significantly better in the worse case than that, if the sensor range of the robot is small. Consider star graphs that consist of many long chains of vertices emanating from the start vertex of the robot. The worst-case travel distance for mapping star graphs is approximately twice their number of vertices since every mapping method needs to traverse each chain almost to its end and then return to the start vertex for all but one chain. Consequently, the worst-case travel distance of depth-first search is minimal or close to minimal (which does not imply that its travel distance for every mapping task is minimal or close to minimal).

The planning times of Greedy Mapping and depth-first search are polynomial in the number of vertices and they both run in real-time. Thus, depth-first search provides a standard that can be used to evaluate the performance of Greedy Mapping. The lower bound on the worst-case travel distance of Greedy Mapping shows that its worst-case travel distance is super-linear in the number of vertices, no matter whether it knows the topology of the graph *a priori*. The worst-case travel distance of Greedy Mapping is thus not minimal. While Greedy Mapping and depth-first search can behave the same in Figure 13, depth-first search cannot traverse the rim repeatedly forward and backward but rather traverses the rim only a second time when it reaches the situation shown in Figure 17. The upper bound on the worst-case travel distance of Greedy Mapping is close to the lower bound (since $\log \log |V|$ is pretty close to one even for large $|V|$) and shows that its worst-case travel distance is not much worse than linear in the number of vertices and thus only slightly suboptimal. The small upper bound is robust as it applies to different versions of Greedy Mapping, regardless of sensor types and sensor ranges, as long as they scan at least the current vertex of the robot.

The small upper bound shows that there is a small penalty to pay for the qualitative advantages of Greedy Mapping since depth-first search does not have the qualitative advantages of Greedy Mapping. However, if the robot can sense information of the topology of the graph or the blockage status of vertices at a distance then it may be possible to map terrain in significantly fewer moves than depth-first search and the argument no longer holds.

Overall, the qualitative advantages of Greedy Mapping, its only slightly worst-case suboptimal travel distance and the robustness of the results for different versions of Greedy Mapping justify its use on robots.

7.7 Implementing Greedy Mapping

We have assumed that the robot is capable of error-free motion and sensing. One way of achieving this, besides robot architectures can compensate for actuator and sensor uncertainty either with either GPS or software, is to let the robots drop short-range transceivers (such as infrared transceivers),

Figure 22: Infrared Transceiver

Figure 23: Greedy Mapping with Infrared Transceivers

as shown in Figure 22 (courtesy of Jonas Svennebring at Lance Sensor System), that can then communicate with each other and the robot and thus constitute a sensor network. The robot always drops a transceiver when it is out of range of any transceiver. Figure 23 shows a possible resulting distribution of transceivers where the filled circles are the transceivers, the small hollow circle is the robot, the arrow is the path of the robot, and the large hollow circles are the communication radii of the transceivers and the robot (with the communication radii of transceivers being larger than the one of the robot). The dashed lines show the communication links. Greedy Mapping can request the unique identification of the transceiver closest to the robot and is then localized not in terms of its coordinates but with respect to its closest transceiver. The transceivers can communicate with each other and perform dynamic programming to determine a closest transceiver at an informative location and the shortest path from the current location of the robot to that transceiver. Thus, Greedy Mapping can be implemented robustly with such a system because the robot is always localized. While executing Greedy Mapping, the robot now drops transceivers, and each transceiver currently costs around one dollar and remains in the environment until it is picked up again. Sometimes the robustness of the resulting system justifies these undesirable properties, for example, when unknown terrain after an earthquake has to be searched for survivors.

7.8 Extensions

Experimental results show that the average-case travel distance of Greedy Mapping is linear in the number of vertices and smaller than that of depth-first search, which is different from the analytical results that show that the worst-case travel distance of Greedy Mapping is super-linear in the number of vertices and worse than that of depth-first search. This difference calls for an average-case analysis instead of a worst-

case analysis. Another open question is whether super-linear travel distance can occur in robot arm configuration space.

8 Greedy Localization

Greedy Localization uses agent-centered search to localize a robot. It maintains the set of hypotheses, the vertices at which the robot could be, because they are consistent with all the observations it has made so far. It always moves the robot on a shortest path so that it reduces the size of the hypothesis set, until this is no longer possible. This corresponds to moving from its current vertex on a shortest path to a closest informative vertex until it cannot reach any informative vertex any longer. An informative vertex is a vertex from which the robot can observe information that allows it to eliminate at least one vertex from the set of hypotheses, that is, the set of vertices consistent with all previous observation.

Figure 9 gives an example of the behavior of Greedy Localization. Initially, it can be in cells E2, E4 and E6. The fastest way to gain information is to move north twice. It could then observe "- - + +", in which case it must be in cell B2. It could also observe "- + + +", in which case it can be in cells B4 and B6. Either way, it has reduced the number of cells it can be in. After the robot executes the moves, it observes "- + + +" and thus can be in cells B4 and B6. The fastest way to gain information is to move east. It could then observe "- + - +", in which case it must be in cell B5. It could also observe "- + - -", in which case it must be in cell B7. Either way, it has reduced the number of cells it can be in. After the robot executes the move, it observes "- + - +" and thus knows that it is in cell B5 and has localized, in this particular case with minimal worst-case travel distance.

Greedy Localization moves after each planning episode from its current vertex to a closest informative vertex, which then becomes and remains uninformative. The number of planning episodes is finite and Greedy Localization is guaranteed to terminate since there are only a finite number of vertices. In fact, the worst-case planning time of Greedy Localization is $O(|V||E| \log |V|)$ with just binary heaps, since there are at most $|V|$ planning episodes and each consists of a shortest path computation. (A more complicated data structure would cost $O(|V|(|E| + |V| \log |V|))$.) After termination, Greedy Localization has reduced the number of vertices that it can be in as much as possible since all vertices that it can reach are now uninformative. This also means that all vertices that it could reach from the start vertex are now uninformative since the graph is undirected, which implies that it could not have reduced the number of vertices more if it had moved differently. Thus, Greedy Localization reduces the number to one and localizes if and only if this is possible from the start vertex.

Greedy Localization can use deterministic search methods because it performs an agent-centered search in exactly all of the deterministic part of the non-deterministic state space, namely exactly up to the point where it executes a non-deterministic move and thus gains more information about its current vertex by reducing the number of possible vertices, as shown with a border in Figure 10.

Greedy Localization is a common-sense robot-navigation method whose behavior is exhibited by the Delayed Planning Architecture (with the viable plan heuristic). The Delayed Planning Architecture was pioneered in robot programming classes at Stanford University and Carnegie Mellon University where Nomad 150 mobile robots had to navigate mazes that were built with three-foot high and forty inch long cardboard walls . Subsequently, [Koenig and Simmons, 1998b] developed Minimax LRTA*, a real-time heuristic search method that generalizes the Delayed Planning Architecture to perform agent-centered searches in a subpart of the deterministic part of the non-deterministic state space. It deals with the problem that it might then not reduce the number of possible vertices and thus needs to avoid infinite cycles in a different way.

Figure 24: NP-Hardness of Localization (1)

8.1 Notation

We define the following notation to be able to state some results formally: Let P be the set of all valid localization plans. Let $T_p(G, v)$ be the travel distance of localization plan $p \in P$ on graph $G = (V, E)$ if it starts at vertex $v \in V$. The travel cost of localization plan $p \in P$ on G then is defined to be $T_p(G) := \max_{v \in V} T_p(G, v)$. Furthermore, the best possible travel cost of any valid localization plan on graph G is defined to be $T(G) := \min_{p \in P} T_p(G)$.

8.2 Complexity Results

It is NP-hard to find valid localization plans with minimal worst-case travel distance for robots with long-range sensors in continuous polygonal terrain. In fact, it is NP-hard to find valid localization plans whose worst-case travel distance is within a factor $c \log |V|$ of minimal in both connected grid graphs and continuous polygonal terrain where localization is possible (for a sufficiently small constant $c > 0$), regardless of sensor types and sensor ranges. Thus, it is already NP-hard to find only near-minimal valid localization plans in a variety of settings. This complexity result implies that a polynomial-time $o(\log |V|)$-factor approximation method for localization is unlikely to exist. A more recent complexity result shows that even a $o(\log^2 |V|)$-factor approximation method is unlikely. On the other hand, there exists a polynomial-time $O(\log^3 |V|)$-factor approximation method for localization on grid graphs. Unfortunately, this approximation method is not (yet) computationally feasible.

We illustrate the NP-hardness result here for a simple case by proving that it is NP-complete to find valid localization plans with minimal worst-case travel distance in (possibly unconnected) grid graphs under the assumptions from Section 3.1. The (unconnected) example grid graphs are good

Figure 25: NP-Hardness of Localization (2)

testbeds for localization methods. Testbeds allow empirical robotics researchers to evaluate their robot-navigation methods, communicate performance results of their methods to others, interpret published performance results of others more easily, and compare their methods against these performance results. In particular, testbeds should include cases that are not too easy to solve because otherwise robot-navigation methods would appear to be more efficient than they actually are. The localization problem is clearly in NP since valid localization plans with minimal worst-case travel distance can be encoded in polynomial length (for example, as decision trees). Thus, it is in NP to determine whether there exists a valid localization plan that executes no more moves than a given value. The localization problem is also NP-hard as can be shown by reducing set-cover problems to localization problems. Set-cover problems consist of a number of elements and a number of sets containing these elements. A set cover is a subset of the sets so that every element is contained in at least one of the sets. It is NP-hard to find a set cover whose number of sets is minimal or even within a log-factor of minimal. We now explain how to construct a localization problem in polynomial time from a given set-cover problem. Figure 24, for example, shows a set-cover problem with 5 elements $e_1 \ldots e_5$ and three sets $S_1 \ldots S_3$, and Figure 25 shows the corresponding localization problem. There is one separate corridor environment for each element and one additional corridor environment for a fictitious element e_0 that is not part of any set. Each corridor environment consists of a number of horizontal corridors that are joined at their east ends by a vertical corridor. There is one horizontal corridor for each set. All horizontal corridors are equally long, with the following exception: Consider a corridor in a corridor environment so that the element that the corridor environment corresponds to is part of the set that the corridor corresponds to. In this case, the corridor is shortened by a number of cells that corresponds to the index of the element that the corridor environment corresponds to. For example, corridor S_2 in corridor environment e_4 is shortened by four cells, while corridor S_2 in corridor environment e_1 is not shortened. The robot starts in the north-east corner of corridor environment e_0. It observes "- + + -" and thus knows that it is in the north-east corner of one of the corridor environments. It is localized once it knows which corridor environment it is in. The corridor environments are constructed in a way so that the robot has to visit at least all the horizontal corridors that correspond to some set cover to localize. For example, assume that the robot visits the corridors that correspond to sets S_2 and S_3, which do not cover element e_1. It then cannot distinguish whether it is in corridor environments e_1 or e_0 (which is the reason for introducing e_0). Since the robot wants to localize with a minimum travel distance, it visits exactly the corridors that correspond to a minimum set cover, which concludes the proof. This proof shows that it is NP-hard to determine whether $T(G) \leq T$ for a given (possibly unconnected) graph G and constant T. A much more complex (connected) version of the example graphs can then be used to prove that there exists a (small) constant $c > 0$ such that it is NP-hard to determine whether there exists a valid localization plan $p \in P$ with $T_p(G) \leq (c \log |V|)T(G)$ for a given graph $G = (V, E)$ even if the graph is connected and localization is possible. The example graphs are unfortunately too complex to yield good testbeds.

8.3 Alternative Approach

Depth-first search can also be used to localize a robot although it is not directed at eliminating possible vertices quickly. It uses depth-first search to acquire the component of the graph that the robot is in, as discussed as an alternative to Greedy Mapping. Then, it determines which of the connected components of the known graph is identical to the acquired graph. The robot is localized if exactly one match is found. Otherwise localization is impossible.

8.4 Empirical Travel Distance

Experimental results show that the average-case travel distance of Greedy Localization tends to be very small both for grid graphs with random obstacles and maze-like grid graphs whose corridors were created by a randomized depth-first search with some additional cells being made unblocked. On the other hand, Since Greedy Localization and depth-first search run in polynomial time and finding valid localization plans with worst-case minimal travel distance is NP-hard, they cannot always find plans with minimal worst-case travel distance.

8.5 Lower Bound on Worst-Case Travel Distance

We now analyze the worst-case travel distance of Greedy Localization as a function of the number of vertices, where the worst case is taken over all graphs of the same number of vertices (perhaps with the restriction that localization is possible), all possible start vertics of the robot, and all tie-breaking strategies used to decide which one of several equally close informative vertices to visit next. More generally, let a be a localization method that (depending on the tie-breaking strategy used to decide between seemingly equally good navigation choices) can produce the set of localization plans $a(G) \subseteq P$ for a given graph G. The worst-case travel distance of localization method a on graphs as a function of their number of vertices n then is defined to be $\max_{G=(V,E):|V|=n} \max_{p \in a(G)} T_p(G)$.

Figure 27: Performance Ratio of Greedy Localization (1)

Figure 26: Lower Bound on Greedy Localization

A lower bound on the worst-case travel distance of Greedy Localization can be established by example. It is easy to see that the worst-case travel distance of Greedy Localization is at least linear in the number of vertices even on planar graphs where localization is possible since it needs to visit about half the vertices at least once if it starts in the center of a chain. We now describe a larger lower bound by presenting a planar (but unconnected) example graph where localization is possible. Consider the base graph shown in Figure 26, which is similar to the example graph from Figure 32 that misleads Planning with the Freespace Assumption. The example graph consists of an exact replica of the base graph and, for each tip vertex, a separate replica of the base graph with that tip vertex deleted. The robot starts in one replica of the graph as shown. It immediately knows from its observation that it is at the left end of the rim of some replica. However, to know which replica it is in, it must check the presence or absence of each tip vertex. Otherwise it can not distinguish the exact replica from a replica whose deleted tip vertex has not been checked. Thus, Greedy Localization visits the spokes in order of ascending length, thus traversing the lengthy rim many times, resulting in Theorem 3.

Theorem 3 *The worst-case travel distance of Greedy Localization is* $\Omega(\frac{|V|\log|V|}{\log\log|V|})$ *moves even on planar (but possibly unconnected) graphs* $G = (V, E)$ *for which localization is possible, where the robot's map may be larger than* G.

The lower bound is not smaller on (possibly unconnected) grid graphs, which can be proved by transforming the example graph into a grid graph, similar to the transformation of the example graph for Greedy Mapping into a grid graph (which is simpler than converting the example graph for Planning with the Freespace Assumption into a grid graph since there is no goal vertex that connects to all tip vertices).

This lower bound can be adapted easily to other robot capabilities. For example, if the robot has long-range sensors, one can add "twists" at the end of the spokes (at the cost of at most a constant factor increase in the number of vertices) so that the robot cannot observe the presence or absence of the tip vertices from the rim and is thus forced to move along the spokes.

8.6 Upper Bound on Worst-Case Travel Distance

Like Greedy Mapping, and by the same argument, Greedy Localization has a worst-case travel distance of at most $|V|^2$ moves, under the base model assumptions. Also, the upper bound was decreased first to $O(|V|^{3/2})$ moves and then to the one given in the Theorem 4.

Figure 28: Performance Ratio of Greedy Localization (2)

Theorem 4 *Under the base model, the worst-case travel distance of Greedy Localization is* $O(|V|\log|V|)$ *moves on arbitrary graphs* $G = (V, E)$.

Proof: If one defines an informative vertex to be unmarked, then Greedy Localization is a greedy agent-centered search method as defined earlier in the context of Greedy Mapping. Lemma 1 then states that its worst-case travel distance is at most $|V| + 2|V|\ln|V|$ moves on graphs $G = (V, E)$. ■

The upper bound of Theorem 4 applies to wide range of situations, including different sensor ranges of the robot, because it relies only on the base model assumptions, which hold for many scenarios, as we have shown.

8.7 Performance Ratio

We now analyze the performance ratio of Greedy Localization, the standard measure of algorithm analysis applied to robot-navigation tasks. It compares the worst-case travel distance of a localization method, such as Greedy Localization, on a given graph against the best possible worst-case travel distance of any localization method that also does not know the location of the robot *a priori*. More generally, let a be a localization method that (depending on the tie-breaking strategy used to decide between seemingly equally good navigation choices) can produce the set of localization plans $a(G) \subseteq P$ for a given graph G. The performance ratio of localization method a on graphs as a function of their number of vertices n then is defined to be $\max_{G=(V,E):\,|V|=n} \max_{p\in a(G)} \frac{T_p(G)}{T(G)}$.

The NP-hardness result shows that there is no polynomial-time localization method with performance ratio less than $c\log|V|$ for a sufficiently small constant $c > 0$. The (unconnected) example grid graph from Figure 27 shows that Greedy Localization can have an arbitrarily bad performance ratio if localization is not possible. One could announce right away that localization is impossible but Greedy Localization executes $|V|/2 - 3$ moves, reducing the number of possible cells by one during each move and only eventually realizing that it cannot distinguish between the remaining two cells. The lower bound on the performance ratio in case localization is

possible was decreased from $\Omega(\sqrt[3]{|V|})$ for a corridor-like example grid graph to the one given in the following theorem for the room-like example grid graph from Figure 28. In both cases, Greedy Localization has a large performance ratio because there may exist a very short "signature" which allows the robot to localize but is in a different direction from other informative (but not completely informative) vertices.

Theorem 5 *The performance ratio of Greedy Localization is* $\Omega(\frac{|V|}{\log |V|})$ *even on grid graphs* $G = (V, E)$ *where localization is possible.*

Proof: Consider grid graphs of the kind shown in Figure 28. They have size $(n \log n + n) \times 5$, where $n \geq 8$ is a power of two. They consist of a row of n blocks of size $(\log n + 1) \times 5$ each. The north-most row of the kth block contains a "signature" that encodes k in binary form, which needs $\log n$ bits. The signature is in form of a pattern of blocked and unblocked cells, followed by a separator that consists of a column of two blocked cells. The remainder of the block consists of unblocked cells. Clearly, the grid graph is connected and has between $4(n \log n + n)$ and $5(n \log n + n)$ unblocked cells. The robot can localize from anywhere in at most $O(1) + 2 \log n$ moves by reading one signature along the north-most row. Thus, the minimal worst-case travel distance to localization is $O(\log n)$ moves. However, if Greedy Localization starts in the cell marked in the figure, then it can reduce the number of possible cells by one in only one move, by moving either west or east. If it moves east, it continues to move east, reducing the number of possible vertices by one during each move. Then, the travel distance of Greedy Localization is $\Omega(n \log n)$ moves. It follows that the performance ratio of Greedy Localization is $\Omega(n \log n / \log n) = \Omega(n) = \Omega(|V| / \log |V|)$. ■

Depth-first search can move in the same way as Greedy Localization on the example grid graph. The performance ratio of Theorem 5 thus applies to depth-first search as well. This lower bound is tight for depth-first search on grid graphs $G = (V, E)$ where localization is possible, for the following reason: The worst-case travel distance of depth-first search is $O(|V|)$ moves. On the other hand, if the robot can localize with a worst-case travel distance of x moves, then it can experience at most $(16)8^x$ different sequences of observations and thus can distinguish at most among this many cells. Thus, it must be the case that $|V| \leq (16)8^x$ and therefore $x = \Omega(\log |V|)$. Put together, the performance ratio of depth-first search is $O(|V| / \log |V|)$.

Theorem 5 also holds for robots with long-range sensors. Furthermore, a similar theorem can be proved for continuous polygonal terrain, regardless of sensor types and sensor ranges, although with a slightly different performance ratio since the size of continuous polygonal terrain must be measured differently from the size of graphs.

8.8 Discussion of Results

The interpretation of the results for Greedy Localization is similar to the interpretation of the results for Greedy Mapping and Planning with the Freespace Assumption. Localization with only near-minimal worst-case travel distance is already NP-hard. The planning times of Greedy Localization and depth-first search are polynomial in the number of unblocked vertices and they both run in real-time. Thus, they both make localization tractable by trading off planning time and travel distance. In fact, the worst-case planning time and travel distance of depth-first search are linear in the number of vertices. No uninformed robot-navigation method can do better than that even on planar graphs where localization is possible, as a chain of vertices shows. The lower bound on the worst-case travel distance of Greedy Localization shows that its worst-case travel distance is slightly super-linear in the number of vertices that are reachable from the start vertex of the robot and thus not minimal. The upper bound is close to the lower bound and shows that its worst-case travel distance is polynomial and not much worst than linear in the number of vertices and thus only slightly suboptimal. Despite the small worst-case travel distances of Greedy Localization and depth-first search, their performance ratios are large. Thus, the localization problems that are responsible for the large performance ratios allow one to localize with a travel distance that is much smaller than linear in the number of vertices.

8.9 Extensions

Robots often suffer from actuator and sensor uncertainty. Empirical robotics researchers have developed robust robot architectures by modeling this uncertainty with probabilities. The robot Xavier, for example, operated very robustly for three years with more than 200 kilometers of travel distance. It used Partially Observable Markov Decision Processes (POMDPs) rather than graphs, where probability distributions over observations rather than (deterministic) observations are associated with vertices and probability distributions over successor vertices rather than (deterministic) successor vertices are associated with moves in vertices. Consequently, the uncertainty of a robot about its current vertex is now represented as a probability distribution over vertices (which is updated using Bayes' rule) rather than a set of possible vertices. POMDP-based ("Markov") Localization considers the robot to be localized when the probability distribution is degenerate (or close to degenerate, that is, has low entropy) rather than when the set of possible states has cardinality one. The localization problem then is to minimize the expected travel distance until the robot is localized from a given initial probability distribution over vertices rather than to minimize the worst-case travel distance until the robot is localized from a given initial set of possible vertices. Empirical robotics researchers suggested and demonstrated a version of Greedy Localization in this case. While Greedy Localization repeatedly makes the robot execute a shortest (deterministic) move sequence that is guaranteed to reduce the number of possible robot cells, this version of Greedy Localization repeatedly makes the robot execute a short move sequence that is guaranteed to decrease the entropy of the probability distribution over the possible cells. The planning time and expected travel distance of this version of Greedy Localization are currently unknown.

Minimizing the expected travel distance for POMDPs is PSPACE-hard in general. However, the POMDPs used in robotics have a regular structure and it might therefore be possible to find localization plans with minimal expected travel

Figure 29: Colored Finite State Automaton

distance in polynomial time, which would be a breakthrough in robotics, especially since the regular structure does not help one to find localization plans with minimal worst-case travel distance, as we explain in the following. Localization on graphs is a special case of finding (adaptive) homing sequences for deterministic finite state automata whose states are colored, a concept from theoretical computer science. A homing sequence is a sequence of moves with the property that the colors of the states observed during its execution uniquely determine the resulting state, and an adaptive homing sequence is a decision tree of moves with the same property. The colored finite state automaton that corresponds to a localization problem on a given graph can be constructed as follows: The vertices of the graph are the states of the finite state automaton, the edges are the actions, and the observations that a robot makes in a vertex determine the color of the corresponding state. Figure 29 shows the colored finite state automaton that corresponds to the graph from Figure 6. Localizing a robot with minimal worst-case travel distance on a given graph then corresponds to finding a shortest adaptive homing sequence in the corresponding finite state automaton. Finding shortest (adaptive) homing sequences on arbitrary colored finite state automata is NP-hard. Finding shortest adaptive homing sequences on colored finite state automata that correspond to grid graphs is not easier since localization on grid graphs with minimal worst-case travel distance is NP-hard as well. Thus, the regular grid structure does not make it easier to find localization plans with minimal worst-case travel distance. It is currently unknown whether this is also the case for finding localization plans with minimal expected travel distance on POMDPs.

9 Planning with the Freespace Assumption

Planning with the Freespace Assumption uses assumption-based planning for goal-directed navigation in unknown terrain that is geometrically embedded. A visual example was given at the beginning of the report, in Figures 2 and 2. The robot optimistically assumes that there are no obstacles, begins to travel on a shortest path to the goal, but encounters an obstacle. The robot updates its knowledge of freespace and replans its path to the goal. In this example, the robot reaches the goal without having detecting all of the obstacles in the space. In particular, it has only detected one edge of the blocked quadrant in workspace when it reaches the goal.

In a discretized problem, the robot knows the vertex-blocked graph $\hat{G}$, its start vertex and the goal vertex but does not know which vertices are blocked. It may or may not know the geometric embedding of the graph. Planning with the Freespace Assumption maintains the partial graph that the robot has learned so far. It always moves the robot on a shortest presumed unblocked path from its current vertex to the goal vertex until it reaches the goal vertex or can no longer find a presumed unblocked path from its current vertex to the goal vertex. A presumed unblocked path is a sequence of neighboring vertices that does not contain vertices that the robot knows to be blocked. The robot makes the optimistic and sometimes incorrect assumption, called the freespace assumption, that vertices with unknown blockage status are unblocked, a popular assumption in robotics, as we have already seen in the context of the parti-game algorithm. The robot moves on the presumed unblocked path toward the goal vertex but immediately repeats the process if it observes a blocked vertex on the planned path. The resulting behavior is equivalent to repeatedly traversing the first edge of a shortest presumed unblocked path from the current vertex to the goal vertex. If the robot reaches the goal vertex, it stops and reports success. If it fails to find a presumed unblocked path from its current vertex to the goal vertex, it stops and reports that the goal vertex cannot be reached from its current vertex and, since the graph is undirected, neither from its start vertex.

Figure 30 gives an example of the behavior of Planning with the Freespace Assumption, using the same symbols as Figure 13. The cross shows the goal cell.

Like Greedy Mapping and Greedy Localization, Planning with the Freespace Assumption requires at most $O(|V|^2)$ moves until termination. The planning time is at most $O|\hat{V}|^3$ because shortest paths are computed on the vertex-blocked graph $\hat{G}$. At termination, the robot has moved to the goal vertex or correctly concluded that this is impossible from its current vertex and, since the graph is undirected, also from its start vertex.

Planning with the Freespace Assumption is a common-sense robot-navigation method that has been used outdoors on NAVLAB II, Carnegie Mellon's unmanned robot HMMWV (high mobility multi-wheeled vehicle) that navigated 1,410 meters to the goal location in an unknown area of flat terrain with sparse mounds of slag as well as trees, bushes, rocks, and debris. As a result of this demonstration, Planning with the Freespace Assumption is now widely used in the DARPA Unmanned Ground Vehicle (UGV) program, for example, on the UGV Demo II vehicles, on Mars Rover prototypes (according to Anthony Stentz), tactical mobile robot prototypes and other military robot prototypes for urban reconnaissance. It has also been used indoors on Nomad 150 mobile robots in robot-programming classes and is the key method of various robot-navigation software, such as the the GRAMMPS planner, a mission planner for multiple robots.

The state space of Planning with the Freespace Assumption is basically identical to the one of Greedy Mapping since the states are again pairs of vertices and partial graphs, namely the current robot vertex and the currently known subgraph. Planning with the Freespace Assumption can use deterministic search methods because it performs assumption-based planning and ignores all outcomes of each action but one,

a) Example Grid

b) Knowledge of the Robot

Figure 30: Planning with the Freespace Assumption

which makes the non-deterministic state space deterministic. This allows it to either follow the path successfully (in case the actual outcomes of all executed actions were not ignored) or gain information about the graph (in case the actual outcome of an executed action was ignored due to an unknown blockage, in which case the incorrect assumption is corrected). Planning with the Freespace Assumption has to re-plan frequently. It can calculate the shortest presumed unblocked paths efficiently with incremental heuristic search methods, such as Dynamic A* or the simpler D* Lite. The computational advantage of incremental heuristic search methods over search from scratch is much larger for Planning with the Freespace Assumption than Greedy Mapping, and Dynamic A* has - in fact - been developed in the context of Planning with the Freespace Assumption, which explains why this robot-navigation method is sometimes also called Dynamic A*.

9.1 Alternative Approach

Depth-first search can also be used for goal-directed navigation in unknown terrain, basically as described in the context of mapping, until the robot has either identified a path from its current vertex to the goal vertex or mapped the graph completely without identifying such a path. However, the travel distance of depth-first search tends to be much larger in practice than the travel distance of Planning with the Freespace Assumption since it does not attempt to move the robot in the direction of the goal vertex. Furthermore, Planning with the Freespace Assumption has several of the advantages over depth-first search that were discussed in the context of Greedy Mapping. Another advantage of Planning with the Freespace Assumption over depth-first search is that Planning with the Freespace Assumption reduces the travel distance of the robot over time when it solves several robot-navigation tasks on the same graph until it eventually moves along a shortest path to the goal vertex. This is so because the freespace assumption makes the robot explore unknown parts of the graph that might result in the discovery of shortcuts.

Figure 31: Implementation of Greedy Mapping

9.2 Empirical Travel Distance

Planning with the Freespace Assumption can be expected to result in short travel distances if the obstacle density is small and the freespace assumption thus is approximately satisfied, which is often the case in practice. It also tends to result in short travel distances if the freespace assumption is not satisfied. For example, experimental results show that the average-case travel distance of Planning with the Freespace Assumption tends to be approximately linear in the number of vertices for maze-like grid graphs whose corridors were created by a randomized depth-first search with some additional cells being made unblocked. The worst-case travel distance of Planning with the Freespace Assumption is not necessarily minimal due to its greedy nature.

9.3 Lower Bound on Worst-Case Travel Distance

We now analyze the worst-case travel distance of Planning with the Freespace Assumption as a function of the number of vertices of the graphs, where the worst case is taken over all graphs of the same number of (unblocked) vertices, the start vertex of the robot, the goal vertex and the tie-breaking strategy used to decide which one of several equally short presumed unblocked paths to move along.

It is easy to see that the worst-case travel distance of Planning with the Freespace Assumption is at least linear in the number of vertices even on planar graphs since it needs to visit every vertex at least once on a chain that has the start vertex of the robot on one end and the goal vertex on the other. We now describe a larger lower bound. We first describe a general result about the relationship of Planning with the Freespace Assumption and Greedy Mapping that can be used to prove the lower bound.

We show that Greedy Mapping can be implemented with Planning with the Freespace Assumption. Assume that the "Closest Unvisited Vertex" version of Greedy Mapping

knows the topology of a given vertex-blocked graph G *a priori* but not which vertices are blocked. Construct a modification G' of graph G as follows: Add a new unblocked vertex to graph G and declare it to be the goal vertex. Connect each vertex of graph G via a new blocked vertex each to the goal vertex. Leave the start vertex of the robot unchanged. Figure 31 gives an example of how the graph from Figure 6 is transformed. Every path taken by Greedy Mapping on graph G can also be taken by Planning with the Freespace Assumption on graph G' and vice versa. Intuitively, Planning with the Freespace Assumption knows that a blocked vertex is blocked if and only if it has already visited a vertex neighboring it. Thus, the shortest presumed unblocked path from the current vertex of the robot to the goal vertex is always a shortest path from the current vertex of the robot to a closest unvisited vertex and from there via two edges to the goal vertex. Planning with the Freespace Assumption traverses this path until it reaches a closest unvisited vertex, only to find out that the next vertex on the path to the goal vertex is blocked. Thus, Planning with the Freespace Assumption repeatedly moves from its current vertex to a closest unvisited vertex on graph G' and thus behaves like Greedy Mapping on graph G, as illustrated in Figure 31. The opposite also holds. For a given path of Greedy Mapping on graph G, one could make the vertex that connects the vertex last visited by Greedy Mapping to the goal vertex unblocked. This does not change the path of Planning with the Freespace Assumption but would allow the robot to reach the goal vertex.

Recall that incremental heuristic search methods were first developed to speed up Planning with the Freespace Assumption. Since Greedy Mapping can be implemented with Planning with the Freespace Assumption, it is immediately obvious that they also have the potential to speed up Greedy Mapping. More importantly, the relationship between Greedy Mapping and Planning with the Freespace Assumption relates the worst-case travel distances of both greedy on-line robot-navigation methods: A lower bound on the worst-case travel distance of Greedy Mapping implies a lower bound on the worst-case travel distance of Planning with the Freespace Assumption, and an upper bound on the worst-case travel distance of Planning with the Freespace Assumption implies an upper bound on the worst-case travel distance of Greedy Mapping. These bounds are identical in big-O notation since graph G' has twice the number of vertices (plus one) than graph G and the same number of unblocked vertices (plus one) as graph G. Furthermore, the graph that results from transforming the example graph from Figure 17 is planar since the example graph is outer-planar, that is, planar with no interior vertices. Consequently, the Theorem 6 follows directly from Theorem 1.

Theorem 6 *The worst-case travel distance of Planning with the Freespace Assumption is* $\Omega(\frac{|V|\log|V|}{\log\log|V|})$ *moves even on planar graphs* $G = (V, E)$.

Figure 32 shows a simplification of the graph that results from transforming the example graph from Figure 17. It basically results in the same behavior of Planning with the Freespace Assumption, fooling it into traversing the lengthy rim many times. Unfortunately, the example graph is somewhat unrealistic and the worst-case travel distance of Planning with the Freespace Assumption could thus be smaller than the lower bound on more realistic graphs. The lower bound is not smaller on grid graphs, which can again be proved basically by transforming the example graph into a grid graph. However, the folding is much more complicated for the example graph for Planning with the Freespace Assumption than for Greedy Mapping because the goal vertex must be simultaneously adjacent to the ends of many spokes of greatly different lengths, which moreover are placed at great distances from each other. In a grid, on the other hand, each cell is adjacent to at most four other cells, and the distance between two adjacent cells is always one. We use several ideas to modify the graph topology to be able to embed it into a grid. A rough conceptual sketch of these ideas is shown in Figure 33 and explained in the following. Overall, the construction is rather complex and therefore does not yield good testbeds.

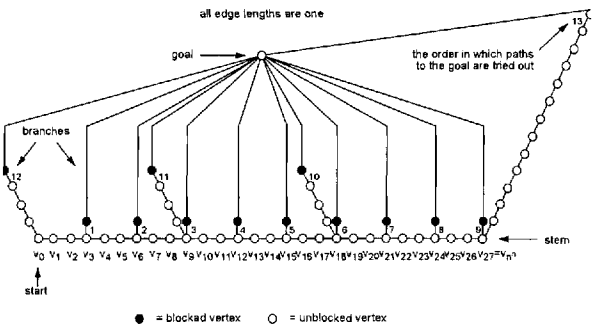

Figure 32: Lower Bound on Planning with the Freespace Assumption

1. Attach each spoke at a separate vertex to the rim (Figure 33 a). This eliminates the problem of a vertex on the rim being adjacent to too many other vertices. As long as longer class spokes are spaced far enough apart, the robot is still fooled into repeatedly traversing the rim.
2. Remove the very short spokes (Figure 33 b). We have to place the goal vertex at some distance D from the rim,

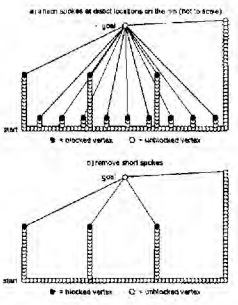

Figure 33: Steps of the Transformation

and we thus cannot construct spokes of length less than D.

3. Move the spokes physically closer together, but maintain their distances from each other along the rim. We do this by "squeezing" the rim into an accordion shape (Figure 33 c).
4. Redesign the spokes so that they all have the same physical height, while maintaining their original lengths (Figure 33 c). In particular, build a pair of blocked walls of the same height, with some space between them. Put a twisty path of the appropriate length in between the walls.
5. Once the spokes are fairly close and of equal height, bend the rim into part of a circular arc, pushing the tip vertices of the spokes together toward the goal vertex (Figure 33 d). It is not possible to squeeze too many distinct vertices into a small area on a grid, but this problem is solved by blocking the paths to the goal vertex a bit before the goal vertex.

9.4 Upper Bound on Worst-Case Travel Distance

It is easy to see that the worst-case travel distance of Planning with the Freespace Assumption is at most $|V|^2$ moves. Since the robot always moves after each planning episode along a shortest presumed unblocked path, it reaches the goal vertex or discovers a blocked vertex on the path that it did not know about after at most $|V|$ moves. Since there are only $|V|$ vertices, there are at most $|V|$ planning episodes, resulting in the upper bound. The upper bound was decreased first to $O(|V|^{3/2})$ moves and then to the one given in the Theorem 7.

Theorem 7 *The worst-case travel distance of Planning with the Freespace Assumption is $O(|V| \log^2 |V|)$ moves on arbitrary graphs $G = (V, E)$ and $O(|V| \log |V|)$ moves on planar graphs $G = (V, E)$ (including grid graphs).*

As argued above, Greedy Mapping can be implemented with Planning with the Freespace Assumption. An upper bound on the worst-case travel distance of Planning with the Freespace Assumption thus implies the same big-O upper bound on the worst-case travel distance of Greedy Mapping. However, the transformed graph is not guaranteed to be planar even if the graph itself is planar. Thus, Theorem 7 implies only that the worst-case travel distance of Greedy Mapping is $O(|V| \log^2 |V|)$ moves on arbitrary graphs, and Theorem 2 provides a tighter upper bound.

The upper bound of Theorem 7 holds for the base model, and therefore applies to many scenarios, including those with different sensor ranges.

9.5 Discussion of the Results

The interpretation of the results for Planning with the Freespace Assumption is similar to the interpretation of the results for Greedy Mapping, given that we made similar assumptions and the upper and lower bounds on the worst-case travel distances of both robot-navigation methods are similar. Again, the worst-case travel distance of depth-first search is at most twice the number of vertices, and no uninformed robot-navigation method in unknown terrain can do significantly better than that if the sensor range of the robot is small. To see that the worst-case travel distance of robot-navigation methods that replace Planning with the Freespace Assumption is at least twice the number of vertices minus a small constant consider the transformation of star graphs. Furthermore, the planning times of Planning with the Freespace Assumption and depth-first search are both polynomial in the number of vertices and they both run in real-time. The lower bound on the worst-case travel distance of Planning with the Freespace Assumption shows that its worst-case travel distance is superlinear in the number of vertices. The worst-case travel distance of Planning with the Freespace Assumption thus is not minimal. The upper bound on the worst-case travel distance of Planning with the Freespace Assumption is close to the lower bound and shows that its worst-case travel distance is not much worse than linear in the number of vertices and thus only slightly suboptimal despite its qualitative advantages.

10 Related Work

The parti-game algorithm is due to [Moore and Atkeson, 1995]. Visibility graphs are derived in [Latombe, 1991]. Information about probabilistic maps (PRMs) is found in [?], and RRTs are described in [?]. [?] illustrates that the obstacles in configuration space can have complex shapes even

if the obstacles in the workspace have very simple shapes. Voronoi diagrams are described in [Latombe, 1991]. We recommend [?] and [?] as further reading on how to discretize freespace efficiently and then find shortest paths for motion-planning problems in known workspaces.

The accuracy tests of sensing and actuation by the mobile Nomad 150 were performed by [Nourbakhsh, 1996]. Hierarchical navigation architectures are described in [Lumelsky, 1987; D. Kortenkamp and Murphy, 1998]. Probabilistic methods at the lower level of these architectures are found in [Thrun, 2000]. The picture of the mobile robot in figure 5 is from [Simmons *et al.*, 2001]. Robot arms are discussed in [Rao *et al.*, 1993]. [Schapire, 1992] analyzes a very early version of SLAM, and also provides an example of recognition without unique vertex identifiers.

Autonomous map acquisition of complex domains is described in [Moore and Atkeson, 1995]. Motivation for the kidnapped robot problem is given in [Cox, 1997; Wang, 1991]. Complete AND-OR searches are described in [Nourbakhsh, 1997; Koenig, 2001b], and their intractability is is discussed in [Genesereth and Nourbakhsh, 1993]. More general results on the difficulty of planning with incomplete information are found in [Littman, 1994; Madani *et al.*, 1999]. Agent-centered search is described in [Korf, 1990],[Ishida, 1992], [Nourbakhsh, 1997],and [Koenig, 2001a]. More general methods are presented in [Russell and Wefald, 1991]. Envelope planning is found in [Dean *et al.*, 1995]. For information on sensor-based planning see [Choset and Burdick, 1994].

Greedy mapping and its use on various mobile robots are described in [Koenig *et al.*, 2001],[Thrun *et al.*, 1998], and [Romero *et al.*, 2001]. A* has been used for decades, [Hart *et al.*, 1968; Pearl, 1984]; dynamic A* is introduced by [Stentz, 1995], and the simpler D* Lite by [Koenig and Likhachev, 2003; 2005]. The relationship between A* and incremental search is explained in [Koenig *et al.*, 2004] for the general case and in [Likhachev and Koenig, 2002] in the context of Greedy Mapping. A subsumption architecture is described in [Brooks, 1986]. Robustness of search methods is discussed in [Agre and Chapman, 1987]. Cooperative terrain mapping methods have been developed in [Singh and Fujimura, 1993; Burgard *et al.*, 2000; Simmons *et al.*, 1997]. Modifications of depth first search are explored in [Wagner *et al.*, 1999].

The empirical travel distances of GM on random grid graphs described here are reported in [Tovey and Koenig, 2003]. The example graph for Greedy Mapping is from [Koenig *et al.*, 2003]. It is a variation of a graph in [Koenig and Smirnov, 1996]. Theorem 1 is taken from [Koenig *et al.*, 2003]; Lemma 1 and Theorems 3 and 2 are taken from [Tovey and Koenig, 2003; Tovey *et al.*, 2006]. The $O(|V|^{3/2})$ bound on greedy mapping was obtained in [Koenig *et al.*, 2001]. [Zlot *et al.*, 2002] uses the upper bounds of theorem 2 to justify the empirical use of Greedy Mapping. The proof for the "Closest Unscanned Vertex with Re-Planning" variant of greedy mapping can be found in [Tovey and Koenig, 2003].

Sensor networks are defined in [Batalin and Sukhatme, 2004]. The travel distance of mapping has been studied in the theoretical robotics and computer science literature principally with respect to the competitive ratio criterion [Sleator and Tarjan, 1985]. [Deng *et al.*, 1998] found the first method (greedy, but not equivalent to Greedy Mapping) with $O(1)$ competitive ratio for rectilinear polygons; and [Hoffman *et al.*, 1997] found one for the more general case of simple polygons. [Albers and Henzinger, 2000; Deng and Papadimitriou, 1990] and references therein study mapping unoriented graphs.

[Cormen *et al.*, 1990] is one of several superb books on data structures and algorithms, including binary heaps and more sophisticated structures such as Fibonacci heaps.

Greedy localization, including the empirical results described here, is studied in [Tovey and Koenig, 2000]. The continuous polygonal hypothesis generation algorithm originates with [Guibas *et al.*, 1992]. See [Dudek *et al.*, 1995] for hypothesis elimination. The delayed planning architecture is described in [Nourbakhsh, 1996], and [Koenig and Simmons, 1998b] developed Minimax LRTA*.

[Dudek *et al.*, 1995] were the first to prove that it is NP-hard to find valid localization plans with minimal worst-case travel distance for robots with long-range sensors in continuous polygonal terrain. The log approximation hardness for both disconnected and connected graphs is proved in [Tovey *et al.*, 2006], by reduction from the set cover result of [Lund and Yannakakis, 1994]. The $\log^3$ approximation algorithm, and a $\log^2$ approximation hardness result, are both found in [?]. Testbeds are discussed in [Hanks *et al.*, 1993]. Theorem 5 and the folded example graph for greedy localization can be found in [Tovey *et al.*, 2006]. The $O(|V|^{3/2})$ bound on greedy localization appeared in [Tovey and Koenig, 2000]; the tighter bound of Theorem 4 is from [Mudgal *et al.*, 2004]. The early bounds on worst-case performance ratio appeared in [Tovey and Koenig, 2000].

The Xavier POMDP architecture and its three year performance is described in [Koenig and Simmons, 1998a]. [Fox *et al.*, 1998] gives an entropy-decrease approach to localization. We recommend [?] as further reading on probabilistic robotics.

[Guibas *et al.*, 1992] gave a geometric polynomial time method for robots with long-range sensors in a polygon to determine the set H of possible locations that are consistent with its initial observation. The travel distance of localization has been studied in the theoretical robotics and computer science literature principally with respect to the competitive ratio criterion, starting with [Papadimitriou and Yannakakis, 1991]. [Dudek *et al.*, 1995] found a best possible online ratio of $2(|H| - 1)$ in continuous polygonal models for robots with slightly limited long-range sensors that can detect the visibility skeleton rather than the visibility polygon. [Kleinberg, 1994] and [Fleischer *et al.*, 2001] analyzed online behavior for a kind of graph, namely geometric trees. [Kleinberg, 1994] gave an $O(|V|^{2/3})$-approximate competitive localization method on geometric trees, which asymptotically improves the "spiral search" technique of [?].

Definitions and complexity results on homing sequences are found in [Kohavi, 1978] and [Schapire, 1992], respectively. The complexity of POMDP expected cost minimization is established in [Papadimitriou and Tsitsiklis, 1987]. Planning with the Freespace Assumption (see [Stentz, 1995;

Figure 34: Greedy On-Line Robot-Navigation Methods

Zelinsky, 1992; Foux *et al.*, 1993; Nourbakhsh and Genesereth, 1996]) is studied in [Koenig *et al.*, 2003]. The HMMWV test is reported in [Stentz and Hebert, 1995]. DARPA-related uses of freespace planning are reported in [Hebert *et al.*, 1999; Matthies *et al.*, 2000; Thayer *et al.*, 2000]; other uses are described in [Nourbakhsh, 1996; Nourbakhsh and Genesereth, 1996],and [Brumitt and Stentz, 1998]. Performance when the freespace assumption does not hold is discussed in [Stentz, 1995; 1997]. The empirical results on random grid graphs are reported in [Koenig *et al.*, 2003]. Complete details of the grid graph transformation for Planning can be found in [Mudgal *et al.*, 2005]. Theorem 6 is proved in [Koenig *et al.*, 2003].

Theorem 7 is found in [Mudgal *et al.*, 2004]; the earlier weaker upper bound is proved in [Koenig *et al.*, 2003; Tovey *et al.*, 2003]. The travel distance of goal-directed navigation in unknown terrain has been studied in the theoretical robotics and computer science literature principally with respect to the competitive ratio criterion. [Blum *et al.*, 1997] study motion in rectilinear polygons; [Icking *et al.*, 1999] study more restricted domains such as street polygons. The most closely related results to the ones we described are the so-called "bug algorithms" [Lumelsky and Stepanov, 1987]. That work considers a mobile robot in the Euclidean plane, with unknown obstacles, moving to a given goal location. The model differs from the one used in this text in some respects: It is continuous and Euclidean, not discrete; the mobile robot uses only local information plus one numerical datum, so it does neither learns a map nor searches all the way to the goal; and the boundary of the region is treated differently. They find two provably correct methods, and close lower and upper bounds on the travel distance in terms of the sum of the Euclidean distance to the goal location and the sum of the perimeter lengths of the obstacles.

11 Conclusions

We gave an overview of our research results whose aim is to provide first steps toward a firm theoretical foundation for greedy on-line robot-navigation methods, including explaining the reasons why empirical robotics researchers use them and the good empirical results that have been reported about them in the experimental literature. We studied Greedy Mapping and Greedy Localization, which perform agent-centered search, and Planning with the Freespace Assumption, which performs assumption-based planning. We explained the qualitative advantages of these polynomial-time robot-navigation methods, stemming from the property that they are reactive to changes in the location of the robot and its knowledge of the terrain since they can re-plan after every move. They plan is real-time and empirically result in short travel distances. Our analysis of their worst-case travel distances showed how these greedy on-line robot-navigation methods can be related using a graph-theoretic framework, as shown in Figure **??**, resulting in similar bounds and similar proofs. We proved that several variants of these robot-navigation methods have small worst-case travel distances that are slightly super-linear and thus not optimal but close to optimal. Consequently, there is a penalty to pay for their qualitative advantages but it is small, which justifies their use on mobile robots.

Acknowledgments

We would like to thank all of our co-authors over the years who originated some of the results presented here and contributed to others: S. Greenberg, W. Halliburton, M. Likhachev, A. Mudgal, R. Simmons and Y. Smirnov. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of NSF, the sponsoring agency of the U.S. government.

References

[Agre and Chapman, 1987] P. Agre and D. Chapman. Pengi: An implementation of a theory of activity. In *Proceedings of the National Conference on Artificial Intelligence*, pages 268–272, 1987.

[Albers and Henzinger, 2000] S. Albers and M. Henzinger. Exploring unknown environments. *SIAM Journal on Computing*, 29(4):1164–1188, 2000.

[Batalin and Sukhatme, 2004] M. Batalin and G. Sukhatme. Coverage, exploration and deployment by a mobile robot and communication network. *Telecommunication Systems Journal: Special Issue on Wireless Sensor Networks*, 26(2):181–196, 2004.

[Blum *et al.*, 1997] A. Blum, P. Raghavan, and B. Schieber. Navigating in unfamiliar geometric terrain. *SIAM Journal on Computing*, 26(1):110–137, 1997.

[Brooks, 1986] R. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2:14–23, 1986.

[Brumitt and Stentz, 1998] B. Brumitt and A. Stentz. GRAMMPS: a generalized mission planner for multiple mobile robots. In *Proceedings of the International Conference on Robotics and Automation*, 1998.

[Burgard *et al.*, 2000] W. Burgard, D. Fox, M. Moors, R. Simmons, and S. Thrun. Collaborative multi-robot exploration. In *Proceedings of the International Conference on Robotics and Automation*, pages 476–481, 2000.

[Choset and Burdick, 1994] H. Choset and J. Burdick. Sensor based planning and nonsmooth analysis. In *Proceedings of the International Conference on Robotics and Automation*, pages 3034–3041, 1994.

[Cormen *et al.*, 1990] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[Cox, 1997] I. Cox. Blanche – an experiment in guidance and navigation of an autonomous robot vehicle. *SIAM Journal on Computing*, 26(1):110–137, 1997.

[D. Kortenkamp and Murphy, 1998] R. Bonasso D. Kortenkamp and R. Murphy. *Xavier: A Robot Navigation Architecture Based on Partially Observable Markov Decision Process Models*. MIT Press, 1998.

[Dean *et al.*, 1995] T. Dean, L. Kaelbling, J. Kirman, and A. Nicholson. Planning under time constraints in stochastic domains. *Artificial Intelligence*, 76(1-2):35–74, 1995.

[Deng and Papadimitriou, 1990] X. Deng and C. Papadimitriou. Exploring an unknown graph. In *Proceedings of the Symposium on Foundations of Computer Science*, pages 355–361, 1990.

[Deng *et al.*, 1998] X. Deng, T. Kameda, and C. Papadimitriou. How to learn an unknown environment i: the rectilinear case. *Journal of the ACM*, 45(2):215–245, 1998.

[Dudek *et al.*, 1995] G. Dudek, K. Romanik, and S. Whitesides. Localizing a robot with minimum travel. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 437–446, 1995.

[Fleischer *et al.*, 2001] R. Fleischer, K. Romanik, S. Schuierer, and G. Trippen. Optimal robot localization in trees. *Information and Computation*, 171:224–247, 2001.

[Foux *et al.*, 1993] G. Foux, M. Heymann, and A. Bruckstein. Two-dimensional robot navigation among unknown stationary polygonal obstacles. *IEEE Transactions on Robotics and Automation*, 9(1):96–102, 1993.

[Fox *et al.*, 1998] D. Fox, W. Burgard, and S. Thrun. Active Markov localization for mobile robots. *Robotics and Autonomous Systems*, 25:195–207, 1998.

[Genesereth and Nourbakhsh, 1993] M. Genesereth and I. Nourbakhsh. Time-saving tips for problem solving with incomplete information. In *Proceedings of the National Conference on Artificial Intelligence*, pages 724–730, 1993.

[Guibas *et al.*, 1992] L. Guibas, R. Botwani, and P. Raghavan. The robot localization problem in two dimensions. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 259–268, 1992.

[Hanks *et al.*, 1993] S. Hanks, M. Pollack, and P. Cohen. Benchmarks, test beds, controlled experimentation, and the design of agent architectures. *Artificial Intelligence Magazine*, 14(4):17–42, 1993.

[Hart *et al.*, 1968] N. Hart, J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on System Science and Cybernetics*, 4(2):100–107, 1968.

[Hebert *et al.*, 1999] M. Hebert, R. McLachlan, and P. Chang. Experiments with driving modes for urban robots. In *Proceedings of the SPIE Mobile Robots*, 1999.

[Hoffman *et al.*, 1997] F. Hoffman, C. Icking, R. Klein, and K. Kriegel. A competitive strategy for learning a polygon. In *Proceedings of the Symposium on Discrete Algorithms*, pages 166–174, 1997.

[Icking *et al.*, 1999] C. Icking, R. Klein, and E. Langetepe. An optimal competitive strategy for walking in streets. In C. Meinel and S. Tison, editors, *Proceedings of the Symposium on Theoretical Aspects of Computer Science*, volume 1563 of *Lecture Notes in Computer Science*, pages 110–120. Springer, 1999.

[Ishida, 1992] T. Ishida. Moving target search with intelligence. In *Proceedings of the National Conference on Artificial Intelligence*, pages 525–532, 1992.

[Kleinberg, 1994] J. Kleinberg. The localization problem for mobile robots. In *Proceedings of the Annual Symposium on Foundations of Computer Science*, pages 521–533, 1994.

[Koenig and Likhachev, 2003] S. Koenig and M. Likhachev. D* lite. In *Proceedings of the National Conference on Artificial Intelligence*, pages 476–483, 2003.

[Koenig and Likhachev, 2005] S. Koenig and M. Likhachev. Fast replanning for navigation in unknown terrain. *Transactions on Robotics*, 2005.

[Koenig and Simmons, 1998a] S. Koenig and R. Simmons. Xavier: A robot navigation architecture based on partially observable markov decision process models. In R. Bonasso D. Kortenkamp and R. Murphy, editors, *Artificial Intelligence Based Mobile Robotics: Case Studies of Successful Robot Systems*, pages 91–122. MIT Press, 1998.

[Koenig and Simmons, 1998b] S. Koenig and R.G. Simmons. Solving robot navigation problems with initial pose uncertainty using real-time heuristic search. In *Proceedings of the International Conference on Artificial Intelligence Planning Systems*, pages 145–153, 1998.

[Koenig and Smirnov, 1996] S. Koenig and Y. Smirnov. Graph learning with a nearest neighbor approach. In *Proceedings of the Conference on Computational Learning Theory (COLT)*, pages 19–28, 1996.

[Koenig *et al.*, 2001] S. Koenig, C. Tovey, and W. Halliburton. Greedy mapping of terrain. In *Proceedings of the International Conference on Robotics and Automation*, pages 3594–3599, 2001.

[Koenig *et al.*, 2003] S. Koenig, Y. Smirnov, and C. Tovey. Performance bounds for planning in unknown terrain. *Artificial Intelligence Journal*, 147(1-2):253–279, 2003.

[Koenig *et al.*, 2004] S. Koenig, M. Likhachev, Y. Liu, and D. Furcy. Incremental heuristic search in artificial intelligence. *Artificial Intelligence Magazine*, 24(2):99–112, 2004.

[Koenig, 2001a] S. Koenig. Agent-centered search. *Artificial Intelligence Magazine*, 22(4):109–131, 2001.

[Koenig, 2001b] S. Koenig. Minimax real-time heuristic search. *Artificial Intelligence*, 129(1–2):165–197, 2001.

[Kohavi, 1978] Z. Kohavi. *Switching and Finite Automata Theory*. McGraw-Hill, second edition, 1978.

[Korf, 1990] R. Korf. Real-time heuristic search. *Artificial Intelligence*, 42(2-3):189–211, 1990.

[Latombe, 1991] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[Likhachev and Koenig, 2002] M. Likhachev and S. Koenig. Incremental replanning for mapping. In *Proceedings of the International Conference on Intelligent Robots and Systems*, pages 667–672, 2002.

[Littman, 1994] M. Littman. Memoryless policies: Theoretical limitations and practical results. In *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, 1994.

[Lumelsky and Stepanov, 1987] V. Lumelsky and A. Stepanov. Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2:403–430, 1987.

[Lumelsky, 1987] V. Lumelsky. Algorithmic and complexity issues of robot motion in an uncertain environment. *Journal of Complexity*, 3:146–182, 1987.

[Lund and Yannakakis, 1994] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM*, 41:960–981, 1994.

[Madani *et al.*, 1999] O. Madani, S. Hanks, and A. Condon. On the undecidability of probabilistic planning and infinite-horizon partially observable Markov decision problems. In *Proceedings of the National Conference on Artificial Intelligence*, pages 541–548, 1999.

[Matthies *et al.*, 2000] L. Matthies, Y. Xiong, R. Hogg, D. Zhu, A. Rankin, B. Kennedy, M. Hebert, R. Maclachlan, C. Won, T. Frost, G. Sukhatme, M. McHenry, and S. Goldberg. A portable, autonomous, urban reconnaissance robot. In *Proceedings of the International Conference on Intelligent Autonomous Systems*, 2000.

[Moore and Atkeson, 1995] A. Moore and C. Atkeson. The parti-game algorithm for variable resolution reinforcement learning in multi-dimensional state spaces. *Machine Learning*, 21(3):199–233, 1995.

[Mudgal *et al.*, 2004] A. Mudgal, C. Tovey, and S. Koenig. Analysis of greedy robot-navigation methods. In *Proceedings of the International Symposium on Artificial Intelligence and Mathematics*, 2004.

[Mudgal *et al.*, 2005] A. Mudgal, C. Tovey, S. Greenberg, and S. Koenig. Bounds on the travel cost of a mars rover prototype search heuristic. *SIAM Journal on Discrete Mathematics*, 2005. (accepted).

[Nourbakhsh and Genesereth, 1996] I. Nourbakhsh and M. Genesereth. Assumptive planning and execution: a simple, working robot architecture. *Autonomous Robots Journal*, 3(1):49–67, 1996.

[Nourbakhsh, 1996] I. Nourbakhsh. *Robot Information Packet*. Distributed at the AAAI-96 Spring Symposium on Planning with Incomplete Information for Robot Problems, 1996.

[Nourbakhsh, 1997] I. Nourbakhsh. *Interleaving Planning and Execution for Autonomous Robots*. Kluwer Academic Publishers, 1997.

[Papadimitriou and Tsitsiklis, 1987] C. Papadimitriou and J. Tsitsiklis. The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.

[Papadimitriou and Yannakakis, 1991] C. Papadimitriou and M. Yannakakis. Shortest paths without a map. *Theoretical Computer Science*, 84(1):127–150, 1991.

[Pearl, 1984] J. Pearl. *Heuristics : Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1984.

[Rao *et al.*, 1993] N. Rao, S. Hareti, W. Shi, and S. Iyengar. Robot navigation in unknown terrains: Introductory survey of non-heuristic algorithms. Technical Report ORNL/TM–12410, Oak Ridge National Laboratory, Oak Ridge (Tennessee), 1993.

[Romero *et al.*, 2001] L. Romero, E. Morales, and E. Sucar. An exploration and navigation approach for indoor mobile robots considering sensor's perceptual limitations. In *Proceedings of the International Conference on Robotics and Automation*, pages 3092–3097, 2001.

[Russell and Wefald, 1991] S. Russell and E. Wefald. *Do the Right Thing – Studies in Limited Rationality*. MIT Press, 1991.

[Schapire, 1992] R. Schapire. *The Design and Analysis of Efficient Learning Algorithms*. MIT Press, 1992.

[Simmons *et al.*, 1997] R. Simmons, D. Apfelbaum, W. Burgard, D. Fox, M. Moors, S. Thrun, and H. Younes. Coordination for multi-robot exploration and mapping. In *Proceedings of the National Conference on Artificial Intelligence*, pages 852–858, 1997.

[Simmons *et al.*, 2001] R. Simmons, R. Goodwin, S. Koenig, J. O'Sullivan, and G. Armstrong. Xavier: An autonomous mobile robot on the web. In R. Goldberg and R. Siegwart, editors, *Beyond Webcams: An Introduction to Online Robots*. MIT Press, 2001.

[Singh and Fujimura, 1993] K. Singh and K. Fujimura. Map making by cooperating mobile robots. In *Proceedings of the International Conference on Robotics and Automation*, pages 254–259, 1993.

[Sleator and Tarjan, 1985] D. Sleator and R. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[Stentz and Hebert, 1995] A. Stentz and M. Hebert. A complete navigation system for goal acquisition in unknown environments. *Autonomous Robots*, 2(2):127–145, 1995.

[Stentz, 1995] A. Stentz. Optimal and efficient path planning for unknown and dynamic environments. *International Journal of Robotics and Automation*, 10(3):89–100, 1995.

[Stentz, 1997] A. Stentz. Best information planning for unknown, uncertain, and changing domains. In S. Koenig,

A. Blum, T. Ishida, and R. Korf, editors, *Proceedings of the AAAI Workshop on On-Line Search*, pages 110–113, 1997. Available as AAAI Technical Report WS-97-10.

[Thayer *et al.*, 2000] S. Thayer, B. Digney, M. Diaz, A. Stentz, B. Nabbe, and M. Hebert. Distributed robotic mapping of extreme environments. In *Proceedings of the SPIE: Mobile Robots XV and Telemanipulator and Telepresence Technologies VII*, volume 4195, 2000.

[Thrun *et al.*, 1998] S. Thrun, A. Bücken, W. Burgard, D. Fox, T. Fröhlinghaus, D. Hennig, T. Hofmann, M. Krell, and T. Schmidt. Map learning and high-speed navigation in RHINO. In D. Kortenkamp, R. Bonasso, and R. Murphy, editors, *Artificial Intelligence Based Mobile Robotics: Case Studies of Successful Robot Systems*, pages 21–52. MIT Press, 1998.

[Thrun, 2000] S. Thrun. Probabilistic algorithms in robotics. *Artificial Intelligence Magazine*, 21(4):93–109, 2000.

[Tovey and Koenig, 2000] C. Tovey and S. Koenig. Gridworlds as testbeds for planning with incomplete information. In *Proceedings of the National Conference on Artificial Intelligence*, pages 819–824, 2000.

[Tovey and Koenig, 2003] C. Tovey and S. Koenig. Improved analysis of greedy mapping. In *Proceedings of the International Conference on Intelligent Robots and Systems*, 2003.

[Tovey *et al.*, 2003] C. Tovey, S. Greenberg, and S. Koenig. Improved analysis of D*. In *Proceedings of the International Conference on Robotics and Automation*, 2003.

[Tovey *et al.*, 2006] C. Tovey, S. Koenig, and D. Vroon. Minimizing localization travel distance. *IEEE Transactions on Robotics*, 2006. (accepted).

[Wagner *et al.*, 1999] I. Wagner, M. Lindenbaum, and A. Bruckstein. Distributed covering by ant robots using evaporating traces. *IEEE Transactions on Robotics and Automation*, 15(5):918–933, 1999.

[Wang, 1991] C. Wang. Location estimation and uncertainty analysis for mobile robots. In I. Cox and G. Wilfong, editors, *Autonomous Robot Vehicles*. Springer, 1991.

[Zelinsky, 1992] A. Zelinsky. A mobile robot exploration algorithm. *IEEE Transactions on Robotics and Automation*, 8(6):707–717, 1992.

[Zlot *et al.*, 2002] R. Zlot, A. Stentz, M. Dias, and S. Thayer. Multi-robot exploration controlled by a market economy. In *International Conference on Robotics and Automation*, pages 3016–3023, 2002.